NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

METHODS FOR QUANTUM CIRCUIT DESIGN AND SIMULATION

by

Jeremy M. Weathers

March 2010

Thesis Advisor:	Ted Huffmire
Second Reader:	James Luscombe

Approved for public release; distribution is unlimited

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
25–3–2010	Master's Thesis	2102-06-01—2104-10-31

4. TITLE AND SUBTITLE

Methods for Quantum Circuit Design and Simulation

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Jeremy M. Weathers

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey, CA 93943

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Department of the Navy

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited

13. SUPPLEMENTARY NOTES

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

14. ABSTRACT

The goal of quantum computing is to harness the unique properties of quantum mechanics to perform computations that are beyond the capabilities of traditional, classical circuits based on transistors. A quantum circuit is an established way of expressing a quantum algorithm. This thesis will describe the development of a workflow for the design and simulation of quantum circuits, as well as the use of this workflow to express and simulate a set of well-known quantum algorithms. This workflow is also used to simulate and analyze a set of well-known quantum error correction schemes.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER *(include area code)*
Unclassified	Unclassified	Unclassified	UU	123	

Standard Form 298 (Rev. 8–98)
Prescribed by ANSI Std. Z39.18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

METHODS FOR QUANTUM CIRCUIT DESIGN AND SIMULATION

Jeremy M. Weathers
Lieutenant, United States Navy
B.S., United States Naval Academy, 2003

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 2010

Author: Jeremy M. Weathers

Approved by: Ted Huffmire
Thesis Advisor

James Luscombe
Second Reader

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The goal of quantum computing is to harness the unique properties of quantum mechanics to perform computations that are beyond the capabilities of traditional, classical circuits based on transistors. A quantum circuit is an established way of expressing a quantum algorithm. This thesis will describe the development of a workflow for the design and simulation of quantum circuits, as well as the use of this workflow to express and simulate a set of well-known quantum algorithms. This workflow is also used to simulate and analyze a set of well-known quantum error correction schemes.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgements

There are several people whom I wish to thank. First and foremost, I would like to thank Professor Ted Huffmire, whose encouragement, guidance, patience, and persistence ensured I completed this thesis. It would have been impossible to do this without your support! Thank you, Ted.

To my Mother and Grandmother, whose mentorship and inspiration continue to inspire me to challenge myself and pursue higher, greater goals throughout life.

Last, but not least, I would like to thank my beautiful, loving and supportive wife, who is a constant shinning star in my life, who never waivers and is always there for me. I would not be the man I am today without you! Thank you Julie!

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

Quantum computers were originally envisioned as a means for simulation of quantum mechanical systems, which cannot be efficiently simulated on classical computers. Since then, there have been many developments in both physics and computer science to overcome the significant technical challenges of implementing large-scale, reliable quantum computers, although many challenges remain. In addition, Moore's Law has resulted in feature sizes on integrated circuits that are less than 30nm, a scale at which quantum effects begin to take hold. Another challenge is the development of quantum algorithms, few of which currently exist. One of the most famous quantum algorithms is Shor's factoring algorithm. Since the strength of public-key cryptography, used to secure data on computer networks, is based on the difficulty of factoring integers, the development of large-scale, reliable quantum computers could allow cryptanalysts to decipher messages that have been encrypted using public-key ciphers.

One of the biggest challenges of realizing large-scale, fault-tolerant, practical quantum computers that are capable of factoring large numbers is the fact that quantum states are short-lived and delicate, making them susceptible to errors and *decoherence*. Fortunately, progress has been made in the development of quantum error correction, which can increase the reliability of quantum circuits by encoding a single logical qubit using several physical qubits.

The design space of quantum circuits is large. Shor's algorithm, for example, can be implemented using different configurations of quantum gates. In addition, there are many different proposed quantum error correction schemes, and each error correction algorithm has several parameters, each of which can be varied depending on the design objectives.

The fundamental idea of quantum computing is that an individual quantum bit can have the values of both zero and one simultaneously, unlike traditional bits that are based on transistors. This concept is called superposition: until the value of the quantum bit is measured, its value is indeterminate and obeys the laws of probability. There is a certain probability of measuring zero and a certain probability of measuring one, but measurement will collapse the quantum system, yielding only zero or one. Since one quantum bit can represent two values (zero and one) simultaneously, N quantum bits can represent 2^N values simultaneously, unlike N classical bits, which can only represent one value at a time. For example, 4 quantum bits in theory

can represent $2^4 = 16$ values simultaneously. The ability to operate on 2^N values simultaneously opens up new opportunities for computer algorithms with greater power than existing algorithms. For example, Shor's algorithm works by applying transformations to N qubits simultaneously, where 2^N is on the order of the number to be factored. However, the cost of error correction puts a damper on any gains provided by quantum parallelism. For example, in theory, the number of quantum bits required to factor a 2048-bit number is on the order of 2048, since 2048 quantum bits can represent 2^{2048} values simultaneously. However, experts in the field predict that factoring a 2048-bit number will actually require on the order of 10^7 quantum gates and 10^6 quantum bits, far more than the theoretical number of quantum bits required.

In addition to superposition, quantum circuits also use entanglement, a quantum mechanics phenomenon in which two particles may be entangled. Measurement of one half of the entangled pair will immediately collapse the two-qubit system such that the measurement of the second half of the pair will yield the same result as the first.

To achieve large-scale, reliable quantum computers will require evaluating the tradeoffs of various quantum error correction schemes and their parameters. This will require design tools that allow the designer to explore the design space by varying the quantum error correction algorithm and the quantum error correction algorithm parameters. Another critical factor in achieving large-scale, reliable quantum computers will be to consider the underlying physical implementation technology. For example, whether or not the quantum circuit is implemented using ion traps or nuclear magnetic resonance will have a huge impact on the reliability of the circuit. Ion traps, for example, have a much longer decoherence time than other types of qubits. In addition to decoherence time, each type of qubit has several other critical parameters, including the time it takes to perform an individual quantum operation by applying a quantum gate, the underlying reliability of the particular physical implementation technology in question, size, cost, etc. Designers need to be able to vary these parameters in order to explore the design space of quantum circuits.

Simulating quantum circuits is limited by the fact that the complexity of the simulation grows exponentially in the number of quantum bits. If simulation could be done efficiently, there would be no need to build a quantum computer in the first place. This thesis will describe the development of a Java-based application for designing quantum circuits. Part of the user interface will display the quantum circuit being designed. The designer will be able to add quantum gates manually using the mouse. The application will also allow the designer to save

the quantum circuit to a file. This file can then be used as input to a separate application that simulates the quantum circuit.

The outline of this paper is as follows: Chapter 2 will discuss the basic theories and concepts that provide the foundation for Quantum Computer Science. Chapter 3 covers quantum circuits, which are an established way of expressing quantum algorithms, along with explanation of superposition and entanglement, which are fundamental building blocks of quantum algorithms. This chapter will also give some mathematical background and examples of quantum gates. Chapter 4 will explain quantum algorithms, which we will express as circuits using the quantum circuit design tool and simulate using the quantum simulator. Chapter 5 discusses the different types of quantum errors that can occur with quantum bits as well as error correction techniques developed to correct these errors. Chapter 6 will discuss several technologies actively being researched for the physical realization of quantum bits. Chapter 7 will explain our quantum circuit design and simulation workflow. Chapter 8 demonstrates our workflow with several examples of quantum algorithms. Chapter 9 will explain the error correction schemes that we analyzed using our workflow. We conclude in Chapter 10 and discuss opportunities for future work and where we believe the field to be headed.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2:
Quantum Computer Science Foundations

Introduction

This chapter presents some foundational concepts of quantum computing on which our workflow for the design and simulation of quantum circuits is based.

2.1 Information is Physical

Information is physical. It obeys the laws of physics and is registered in cells, neurons, brains, DNA, and transistors. In the words of Rolf Landauer, "Computation is inevitably done with real physical degrees of freedom, obeying the laws of physics, and using parts available in our actual physical universe [3]."

2.1.1 Information Theory

The transmission of digital signals in the form of 0s and 1s through a communication channel ultimately requires some form of physical medium to hold the information.

The processing of information has been studied at length as it applies to classical computers [4]. Bennett and Landauer have shown that erasure of information results in dissipation of thermal energy [3, 5, 6]. An example of information erasure is an AND gate, which has two inputs and one output. An AND gate is not reversible because it is impossible to determine the two inputs given the one output. For this reason, an AND gate erases one bit of information. The same is true of an OR gate. However, a NOT gate has only one input and one output, and it is possible to determine the input given the output. Therefore, the NOT gate is reversible and can be used in a quantum computer. A quantum computer can only use reversible gates because interaction between subatomic particles must be symmetrical in time, and running the reaction backwards yields the original state. Information is conserved, and histories are preserved. Information leaves a quantum system upon measurement or observation. Reversible gates were originally designed for classical computers [7].

2.1.2 Quantum Information Theory

Quantum information theory, an information theory based on quantum principles, is an "extension[sion] and a completion [tion] of classical information theory [8]." An analogy can be

made to complex numbers, which extend and complete the real numbers. Quantum information theory "includes quantum generalizations of classical notions such as sources, channels, and codes, as well as two complementary, quantifiable kinds of information—classical information and quantum entanglement [8]." We will explain the distinction between classical bits and quantum bits as well as the distinction between classical gates and quantum gates.

2.2 Classical Computers

What Is a Classical Computer?

Classical computers are based on the laws of classical physics. As manufacturers continue to make transistors and microchips smaller and smaller, it is inevitable that they will cross the threshold from classical physics to quantum physics. Transistors that are mere tens of nanometers in size can now be fabricated. As sizes decrease further, quantum effects will begin to take hold. This push is one factor motivating the evolution of computer history and quantum computer development.

2.3 Quantum Computers

2.3.1 Quantum Computing

Quantum computing makes use of quantum mechanics. Quantum computing is the result of combining traditional computer science, quantum mechanics, and quantum information theory.

2.3.2 What is a Quantum Computers?

If a classical computer is based on classical physics then it would be logical that quantum computers follow the laws of quantum physics.

Physically, quantum computers will not be built like today's classical computers. However, this is not to say that the design principles of classical computers will not influence those of quantum computers. There are many technologies being researched for the physical realization of a quantum computer. Some of these technologies are quantum dots, nuclear magnetic resonance, superconducting junctions and ion traps. Since the experimental realization of a quantum computer is still in its infancy, there is currently no universally accepted technology. Research will determine which technology or technologies are optimal for building computers that can implement useful algorithms.

2.3.3 Implications of a Quantum computer

The performance gains arising from quantum parallelism are one implication. Quantum parallelism can be exploited for the ability to factor large numbers or search huge databases. Factoring numbers is a computationally expensive task for a classical computer. While factoring is computationally expensive, multiplying two factors together is computationally cheap. Asymmetric cryptography leverages this property, since encryption and decryption are based on multiplication, while cryptanalysis requires factoring.

Peter Shor developed a factoring algorithm [9] for use on a quantum computer. The ability to factor large numbers using Shor's factoring algorithm would make it possible to undermine the public-key crypto on which we all rely. Shor's Algorithm has already been experimentally realized using nuclear magnetic resonance technology to factor the number 15 into its prime factors of 3 and 5 [10]. Other experimental realizations of Shor's Algorithm use photonic chips [11, 12]. Another famous quantum algorithm is Grover's algorithm [13] for search.

It is difficult to design quantum algorithms due to the fact that quantum mechanics is not intuitive to algorithm designers. Furthermore, classical algorithm development is hard, and a quantum algorithm must outperform the best known classical algorithm! Quantum algorithms will be re-visited in more detail in Chapter 3.

2.3.4 Classical Computers versus Quantum Computers

Will quantum computers ever replace classical computers? One cannot be absolutely certain of the future, but for now it is safe to say that quantum computers will probably never replace classical computers. Quantum computers will be better at some problems than classical computers, but classical computers will always outperform quantum computers on other problems. Furthermore, quantum computers will be classically controlled.

2.3.5 Challenges

There are numerous obstacles to overcome before realizing a working large-scale quantum computer. A major challenge is that quantum states are short-lived and are susceptible to contamination from interaction with noise in the environment. To overcome this, quantum error correction was devised. However, the cost of quantum error correction is high. Rather than requiring on the order of 2048 qubits to factor a 2048-bit number, it has been estimated that using ion trap technology, it will require 1M qubits and 10M quantum gates due to the overhead of quantum error correction [14].

2.4 Quantum Key Distribution

Now that we have presented some of the basics of quantum computing, we turn to a discussion of quantum key distribution, often abbreviated as QKD. The focus of this thesis is quantum computing; however, it is important to have a general understanding of QKD and how it is distinct from quantum computing. Unlike QKD, which has already been commercialized, quantum computing is still in its infancy.

2.4.1 What is Quantum Key Distribution?

QKD uses quantum theory to create and distribute a secret key over a quantum channel between two parties, which we will call Alice and Bob. Regardless of the protocol, QKD makes use of individual photons of light and either uses the Heisenberg Uncertainty Principle or Quantum Entanglement to distribute a symmetric key between Alice and Bob. The uncertainty principle and entanglement theory are discussed in further detail in Chapter 2. There are many different QKD protocols such as BB84 [15] (being the first protocol), E91 [16], B92 [17] and BB92 [18] to name a few. The strength of QKD lies in the uncertainty principle: anyone eavesdropping on the quantum channel can be detected, as eavesdropping will disturb the quantum state.

2.4.2 What makes QKD work?

QKD implementation requires devices capable of sending and receiving individual photons of light. The two devices need to communicate over a quantum channel. In the case of photons, an optical fiber cable serves as the quantum channel. In a perfect world, in the absence of eavesdropper, Alice would send photons to Bob and establish a quantum key. Once this secret key has been established between Alice and Bob, it can then be used for a classical encryption of a classical communication channel.

At this point the sending and receiving of information from one party to another party is no different that the technology already in place. The information sent would use the necessary protocol required and would be encrypted using classical cryptography based on requirements of the protocol. With QKD, Alice and Bob each have a polarizer, which can be arbitrarily configured to horizontal or rectilinear polarization. Alice chooses the configuration of her polarizer randomly, as does Bob (half the time they will choose the same configuration, and half the time they will choose opposite configurations). If Alice has selected a rectilinear polarization, she can encode zeroes as horizontally polarized photons and ones as vertically polarized photons. After Alice has finished transmitting to Bob, Alice and Bob use a classical communication

channel to communicate the polarization settings they used. If Alice and Bob used the same setting for a particular bit, that bit becomes part of the key. If they used different settings for a particular bit, that bit is discarded [15].

2.4.3 Commercialization and Implementations

Currently there are companies developing quantum devices, such as IDQ (id Quantique) and MagiQ [19]. In 2009, the New Journal of Physics published an article discussing a QKD network in Vienna. This is a real-world commercial application of QKD. This network was designed and implemented by the European project, Secure Communication based on Quantum Cryptography (SECOQC) [20].

2.4.4 Flaws of Implementation

If the physical implementation of QKD is flawed, it may be possible to eavesdrop on the communication. For example, the apparatus may emanate sound or an electromagnetic signal that varies depending on whether a one or a zero is being transmitted. Another drawback is the high cost of single photon sources, single photon detectors, deployment of optical fiber cable optimized to the required wavelengths, and other optical and electronic components. There are also limitations on the distance between sender and receiver based on dissipation, requiring quantum repeaters, which retransmit the incoming signal on the outgoing link. Due to the no-cloning theorem, a true quantum repeater cannot be devised; therefore, the secret data is susceptible inside the repeater device. However, quantum repeaters based on entanglement swapping have been proposed. In additional to optical fiber, researchers are also working on methods to transmit photons through free space (e.g., between satellites and ground stations) [21].

2.5 The Future of Quantum Computers

It is difficult to make predictions about whether or when large-scale, fault-tolerant quantum computers will be built. Moore's Law was predicted to end a long time ago, yet ingenious developers kept finding ways to stretch Moore's Law. Much progress must be made before a large-scale, fault-tolerant quantum computing can be fully realized. This is not to say it cannot be done. Small-scale quantum computes have been built to implement quantum algorithms for small input sizes, as with Shor's Algorithm mentioned above. Given the active research being pursued in this area on both algorithms and physical implementations of quantum bits and the ability of quantum error correction to protect the delicate quantum states from interaction with their environment, large-scale quantum computers may be seen sooner than thought possible (provided that quantum bits can be manufactured with sufficient reliability and at sufficiently low cost).

It is likely that quantum computer implementations will leverage the miniaturization technology and manufacturing infrastructure for the implementation of classical computer chips on silicon wafers (i.e., quantum computers may be built on wafers using similar manufacturing techniques as classical computer chips). Some of the design principles of classical computer chips may also be applicable to quantum computer chips.

CHAPTER 3:
Quantum Circuits

Introduction

Quantum algorithms work by exploiting the properties of quantum parallelism, meaning that N quantum bits can represent 2^Nvalues simultaneously. A function $f(x)$ can be evaluated for all 2^N values of x simultaneously. First, the quantum bits are initialized to a specific value (e.g., 0 or 1). Next, the quantum bits are put into a superposition state using Hadamard gates (H). The concept of superposition is explained in the next section. Next, a transform is applied to this superposition, so that a function can be evaluated for all 2^N input values. At the end of the algorithm, some or all of the quantum bits are measured, collapsing the superposition to a single value, which is the output.

3.1 Superposition and Quantum States

3.1.1 Superposition

The ability of a quantum system to be in a superposition is the advantage of quantum computing. What does it mean to be in a superposition?

In order to understand the *superposition principle* we must recognize what a quantum system is and the use of ket notation (also known as Dirac notation) which is "$|\ \rangle$." Anything inside the ket notation is representative of a quantum system. In computer science the classical bit can either be in the state 0 or 1. In the quantum computing domain a qubit can be in the states $|0\rangle$, $|1\rangle$, or an arbitrary superposition of $|0\rangle$ and $|1\rangle$.

The *superposition principle* states that if $|x\rangle$ and $|y\rangle$ are two states of a quantum system, then any superposition $\alpha\,|x\rangle + \beta\,|y\rangle$ should also be an allowed state of a quantum system where $|\alpha|^2 + |\beta|^2 = 1$. In other words, the probabilities of measuring either state should add up to 1 [22]. The numbers α and β are actually complex numbers and represent the amplitudes of the basis states.

3.1.2 Quantum States

For example, take a quantum system

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}, \text{ where } |\alpha|^2 + |\beta|^2 = 1. \tag{3.1}$$

Based on the definition of the *superposition principle* above, there is a probability associated with measuring the quantum system $|\psi\rangle$. If the system resulting in a $|0\rangle$ measured, then the resultant values of $\begin{bmatrix} \alpha \\ \beta \end{bmatrix}$ will equal $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$. If the system resulting in a $|1\rangle$ measured, then the resultant values of $\begin{bmatrix} \alpha \\ \beta \end{bmatrix}$ will equal $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$.

This can be more generally recognized as:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$
$$|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{3.2}$$

It can be seen that by measuring the system and getting $|0\rangle$, where $\alpha = 1$ and $\beta = 0$, then $|\alpha|^2 + |\beta|^2 = 1$ holds true.

A typical quantum algorithm works as follows:

After the quantum bits are put into a superposition state, a quantum gate is applied to all the qubits that evaluates a function $f(x)$ for all 2^N values of x. This is followed by the application of additional quantum gates to some or all of the qubits, depending on the algorithm. At the end of the algorithm, one or more of the quantum bits is measured. Measurement is a projection of the quantum state of a qubit onto the basis vectors $|0\rangle$ and $|1\rangle$. The measurement is then interpreted appropriately, depending on the algorithm, to get the answer.

Now that we have a fundamental understanding of a single-qubit system, it follows naturally to consider multiple-qubit systems, but first, we must understand tensor products.

3.2 The Tensor Product

As we have seen, a qubit is expressed mathematically as a 2x1 column matrix (also known as a 2x1 column vector). Mathematically expressing multiple-qubit systems requires using the tenor product of matrices. The definition of the tensor product as taken from Chuang and Nielsen's book on Quantum Computation and Quantum Information states: If A is an m-by-n matrix and B is a p-by-q matrix, then $A \otimes B$ is a block matrix of size mp-by-nq, and $a\,\epsilon A$ and $b\,\epsilon B$. Equation (3.3) shows the computation of the tensor product $A \otimes B$. Further expansion of equation 3.3 results in 3.4.

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix} \tag{3.3}$$

$$A \otimes B = \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} & \cdots & a_{11}b_{1q} & \cdots & \cdots & a_{1n}b_{11} & a_{1n}b_{12} & \cdots & a_{1n}b_{1q} \\ a_{11}b_{21} & a_{11}b_{22} & \cdots & a_{11}b_{2q} & \cdots & \cdots & a_{1n}b_{21} & a_{1n}b_{22} & \cdots & a_{1n}b_{2q} \\ \vdots & \vdots & \ddots & \vdots & & & \vdots & \vdots & \ddots & \vdots \\ a_{11}b_{p1} & a_{11}b_{p2} & \cdots & a_{11}b_{pq} & \cdots & \cdots & a_{1n}b_{p1} & a_{1n}b_{p2} & \cdots & a_{1n}b_{pq} \\ \vdots & \vdots & & \vdots & \ddots & & \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots & & \ddots & \vdots & \vdots & & \vdots \\ a_{m1}b_{11} & a_{m1}b_{12} & \cdots & a_{m1}b_{1q} & \cdots & \cdots & a_{mn}b_{11} & a_{mn}b_{12} & \cdots & a_{mn}b_{1q} \\ a_{m1}b_{21} & a_{m1}b_{22} & \cdots & a_{m1}b_{2q} & \cdots & \cdots & a_{mn}b_{21} & a_{mn}b_{22} & \cdots & a_{mn}b_{2q} \\ \vdots & \vdots & \ddots & \vdots & & & \vdots & \vdots & \ddots & \vdots \\ a_{m1}b_{p1} & a_{m1}b_{p2} & \cdots & a_{m1}b_{pq} & \cdots & \cdots & a_{mn}b_{p1} & a_{mn}b_{p2} & \cdots & a_{mn}b_{pq} \end{bmatrix}. \tag{3.4}$$

This is not to be confused with matrix multiplication, which is the mathematical equivalent of applying a quantum gate to a quantum system.

Examples

The following examples show that the tensor product of:

- $|0\rangle$ and $|0\rangle$ is equivalent to $|00\rangle$
- $|0\rangle$ and $|0\rangle$ is equivalent to $|01\rangle$
- $|0\rangle$ and $|0\rangle$ is equivalent to $|10\rangle$
- $|0\rangle$ and $|0\rangle$ is equivalent to $|11\rangle$

The tensor product examples above all represent a 2-qubit quantum system, in which two quantum wires each have one qubit attached. With an understanding of the basis states in equation 3.2, we can go through the following examples:

1. Show that $|0\rangle \otimes |\,0\rangle = |00\rangle$. By re-writing $|0\rangle$ in matrix form, the tensor product can be performed as follows:

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ 0 \begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 1 \cdot 1 \\ 1 \cdot 0 \\ 0 \cdot 1 \\ 0 \cdot 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = |00\rangle \tag{3.5}$$

2. Show that $|0\rangle \otimes |1\rangle = |01\rangle$ Similar to example 1, the matrix form of $|1\rangle$ can be used to perform the tensor product with $|0\rangle$.

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ 0 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 1 \cdot 0 \\ 1 \cdot 1 \\ 0 \cdot 0 \\ 0 \cdot 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = |01\rangle \tag{3.6}$$

3. Show that $|1\rangle \otimes |0\rangle = |10\rangle$

$$\begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ 1 \begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 0 \cdot 1 \\ 0 \cdot 0 \\ 1 \cdot 1 \\ 1 \cdot 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = |10\rangle \tag{3.7}$$

4. Show that $|1\rangle \otimes |1\rangle = |11\rangle$

$$\begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ 1 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 0 \cdot 0 \\ 0 \cdot 1 \\ 1 \cdot 0 \\ 1 \cdot 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = |11\rangle \tag{3.8}$$

In other words, the 2-qubits, $|00\rangle$, can be thought of individually or together; they are mathematically equivalent.

For a 2-qubit system, there are four basis states: $|00\rangle$, $|01\rangle$, $|10\rangle$ and $|11\rangle$. The complex coefficients, or amplitudes, associated with each of these basis states are α_{00}, α_{01}, α_{10} and α_{11}. Therefore,

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle, \text{ where } |\psi\rangle = \begin{bmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{10} \\ \alpha_{11} \end{bmatrix}$$

and $|\alpha_{00}|^2 + |\alpha_{01}|^2 + |\alpha_{10}|^2 + |\alpha_{11}|^2 = 1$. (3.9)

Without any difficult math, if all of the coefficients are equal to $1/2$, it can easily be seen that there is a 25% chance of measuring each basis state of the quantum system $|\psi\rangle$.

3.3 Quantum Gates

Just as classical gates like AND, OR, and NOT are applied to classical bits, quantum gates are applied to quantum bits. Unlike AND and OR gates, which are not reversible (the NOT gate is reversible), quantum gates must be reversible because the interaction of subatomic particles must be symmetrical in time. In other words, running the reaction backwards must yield the same state. Information is conserved, and histories are preserved. Information leaves a quantum system upon measurement or observation. The AND gate, in classical computers, is not reversible because it has two inputs and one output: in general, the two inputs cannot be determined given the one output. Reversible gates, expressed mathematically, are unitary matrices.

3.3.1 PAULI Gates/Matrices

Three common single-qubit gates are expressed mathematically as Pauli matrices, which are 2x2 matrices. A 2x2 quantum gate can be applied to a single quantum bit (a 2x1 column vector). The Pauli matrices are expressed as follows:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$
$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$
$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{3.10}$$

where i represents complex numbers. All three of these gates perform a specific function in quantum computation and information. The gates are often denoted as X, Y and Z, respectively. A fourth gate, the identity matrix (I), is familiar in linear algebra:

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{3.11}$$

The identity matrix is used to represent a situation when no quantum gate has been applied to a quantum bit (see Figure 3.1 and 3.2). This is extremely useful when one quantum bit is not being operated upon while other quantum bits are being operated upon, as seen in equation 3.21.

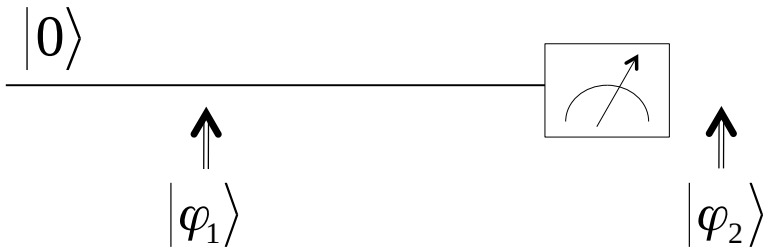


Figure 3.1: Single qubit circuit with no Identity gate

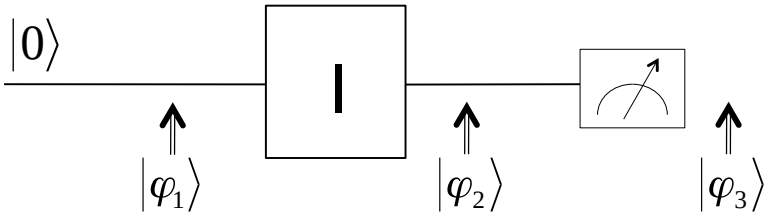


Figure 3.2: Single qubit circuit with an Identity gate

Each of gate a performs specific function within a quantum circuit.

- Purpose/Definition of a X-gate:
  The X gate is an important quantum gate, for it is the equivalent of a NOT gate in classical systems. A NOT gate in a classical circuit allows the input bit to be flipped: if the input is a 1 then the output is a 0, and if the input qubit is a 0 then the output is a 1, as shown in the following mapping:

$$\left.\begin{array}{l} |0\rangle \mapsto |\,1\rangle \\ |1\rangle \mapsto |\,0\rangle \end{array}\right\} \text{flipped basis state}$$

- Purpose/Definition of a Z-gate:
  The Z gate acts on a 1-qubit system by leaving the qubit in the basis state $|0\rangle$ unchanged and performing a sign flip on everything else, as shown in the following mapping:

$$\begin{array}{l} |0\rangle \mapsto |\,0\rangle \quad \text{no state change} \\ |1\rangle \mapsto |-1\rangle \text{ sign flip change} \end{array}$$

3.3.2 Hadamard Gate

The Hadamard gate is a very important quantum gate often used in quantum circuits because of its ability to place qubits in a superposition. In quantum information processing, the Hadamard transformation, more often called Hadamard gate, is a one-qubit operation that transforms either basis state into an equal superposition of both basis states. In other words, there is an equal probability of measuring either basis state. Mathematically, a Hadamard gate is expressed as follows:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{3.12}$$

The following examples show how the Hadamard gate transforms the basis state $|0\rangle$ into an equal superposition of both basis states:

$$H \cdot |0\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \cdot 1 + 1 \cdot 0 \\ 1 \cdot 1 + (-1 \cdot 0) \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{3.13}$$

which is the same as $\begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$. To confirm this we know that $|\alpha|^2 + |\beta|^2$ must equal 1, where $\alpha = \frac{1}{\sqrt{2}}$ and $\beta = \frac{1}{\sqrt{2}}$: $\left|\frac{1}{\sqrt{2}}\right|^2 + \left|\frac{1}{\sqrt{2}}\right|^2 = 1$. Notice that in this case, both α and β are equal to each other. This means that for this qubit there is an equal superposition of $|0\rangle$ and $|1\rangle$; therefore, when the system is measured, there is an equal chance of measuring either $|0\rangle$ and $|1\rangle$.

$$H|0\rangle = \tfrac{1}{\sqrt{2}}|0\rangle + \tfrac{1}{\sqrt{2}}|1\rangle \tag{3.14}$$

$$H|1\rangle = \tfrac{1}{\sqrt{2}}|0\rangle - \tfrac{1}{\sqrt{2}}|1\rangle \tag{3.15}$$

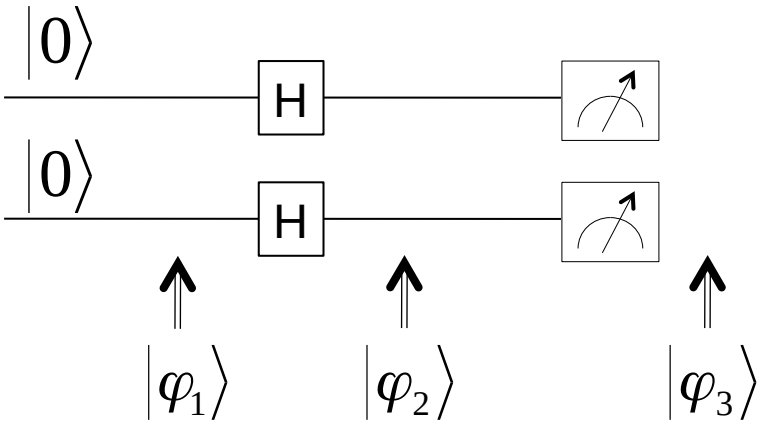


Figure 3.3: Two qubits on two separate quantum wires with two Hadamard gates

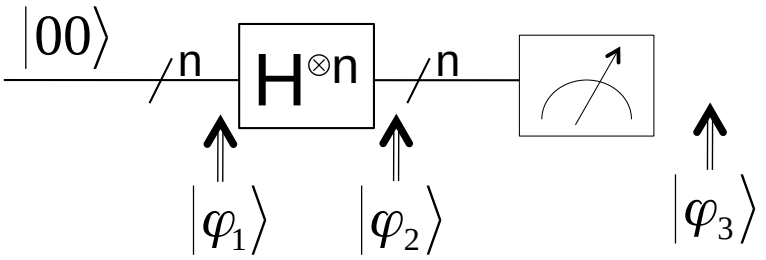


Figure 3.4: Quantum circuit that is equivalent to figure 3.3

Figures 3.3 and 3.4 depict the same quantum circuit. As stated previously, a 2-qubit system comprised of $|0\rangle$ and $|0\rangle$ can be thought of as two 1-qubit systems ($|0\rangle \otimes| 0\rangle$) or one 2-qubit

system ($|00\rangle$); they are equivalent. The key difference between the two circuit diagrams is the understanding of the Hadamard gate. When applying an H-gate to a single qubit, the H-gate is a 2 x 2 matrix. When two H-gates are applied to a 2-qubit system, the two H-gates together form a quantum system that can be mathematically expressed as 4 x 4 matrix, often drawn as $H^{\otimes n}$, where n is the number of H gates simultaneously being applied. Therefore, an n-qubit gate would be expressed as a matrix of size 2^n x 2^n. Mathematically, the equation is expressed as follows:

$$(H^{\otimes 2}) \cdot (|0\rangle \otimes |0\rangle)$$

(3.16)

If we break this equation up into parts:

- Part 1:

$$H^{\otimes 2} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \frac{1}{\sqrt{2}} \cdot \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}$$

(3.17)

- Part 2:

$$|0\rangle \otimes |0\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.18}$$

- Part 3: Put together part 1 and 2

$$(H^{\otimes 2}) \cdot (|0\rangle \otimes |0\rangle) = \tfrac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{3.19}$$

After matrix multiplication, the result is $\frac{1}{2} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \end{bmatrix}$.

Checking the equation $|\alpha_{00}|^2 + |\alpha_{01}|^2 + |\alpha_{10}|^2 + |\alpha_{11}|^2 = 1$, we see that $\frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4} = 1$.

From equation 3.16, which can be written as $H^{\otimes 2} |00\rangle$ or $H \otimes H |00\rangle$, $H^{\otimes 2}$ can be represented as a 4 x 4 matrix. From this it can be shown that the two circuits in Figures 3.3 and 3.4 are equivalent as follows:

$H^{\otimes 2} |00\rangle = (H \otimes H) \cdot |00\rangle$, where $|00\rangle = |0\rangle \otimes |0\rangle$, resulting in $(H \otimes H) \cdot (|0\rangle \otimes |0\rangle)$, demonstrated in equations 3.17, 3.18, and 3.19.

In the next example we will show a 2-qubit system with an H-gate applied to one quantum wire and no gate applied to the second quantum wire. Although no gate is applied to the second

quantum wire, it is equivalent to a circuit with an I-gate on the second wire since mathematically, an Identity matrix has no effect. For this circuit, we only apply the H-gate to one of the two qubits in the system. This circuit can be represented mathematically as:

$$(H \otimes I) \cdot (|0\rangle \otimes |0\rangle), \tag{3.20}$$

where the calculations for $|0\rangle \otimes |0\rangle$ were shown in 3.18; therefore, we will work through the tensor product of H and I (see Figure 3.5).

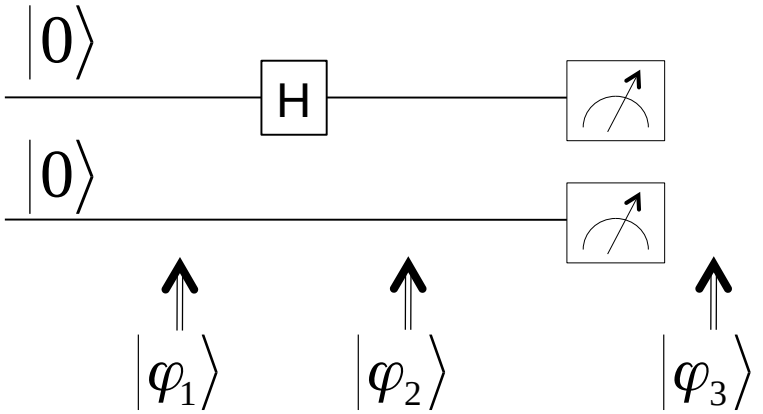


Figure 3.5: Quantum circuit of two qubits, where one qubit has a Hadamard gate applied to it, and the other qubit has no gate applied. Applying no gate is the same as applying an Identity gate.

$$H \otimes I = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \cdot 1 & 1 \cdot 0 & 1 \cdot 1 & 1 \cdot 0 \\ 1 \cdot 0 & 1 \cdot 1 & 1 \cdot 0 & 1 \cdot 1 \\ 1 \cdot 1 & 1 \cdot 0 & -1 \cdot 1 & -1 \cdot 0 \\ 1 \cdot 0 & 1 \cdot 1 & -1 \cdot 0 & -1 \cdot 1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \tag{3.21}$$

We next multiply $H \otimes I$ and $|00\rangle$:

$$(H \otimes I) \cdot |00\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}. \tag{3.22}$$

As shown in Figure 3.5, quantum circuits are read from left to right, but when a quantum circuit is expressed mathematically, the matrices are written from right to left.

3.3.3 Reversible Gates

In general, a distinction between quantum gates and classical gates is that quantum gates are reversible. In a classical computing system, AND and OR gates are not reversible; however, the classical NOT gate is reversible.

> In the 1960s, Rolf Landauer analyzed computational processes and showed that erasing information, as opposed to writing information, is what causes energy loss and heat. This notion has come to be known as Landauer's principle. It has been shown that erasing information is an irreversible, energy-dissipating operation and that if erasing information is the only operation that uses energy, then a computer that is reversible and does not erase would not use energy. [22]

In quantum computing, due to the nature of quantum mechanics, "all operations that are not measurements are reversible [22]." Once a quantum system is measured, the system undergoes collapse, which cannot be reversed. Therefore, because quantum mechanics is reversible, quantum computers must be built with reversible gates [23].

CNOT Gate

The controlled NOT gate is a reversible gate often written as CNOT, which is shown in Figure 3.6. This gate starts with two inputs and yields two outputs. With a CNOT gate, two parts must

be understood, the control bit and the target bit. The control bit (top) is shown as a solid dot, and the target bit (bottom) is represented by the symbol $\oplus$, known as the exclusive-or operator (XOR).

If the control bit input is a $|0\rangle$, then the target bit will be unchanged. However, if the control bit is a $|1\rangle$ then the target bit will flip.

CNOT gate matrix: $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$

Toffoli Gate

The Toffoli gate is very similar to the CNOT gate; it is actually a controlled-controlled NOT gate. The Toffoli gate has two control bits and one target bit.

Toffoli gate matrix: $\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$

Swap Gate

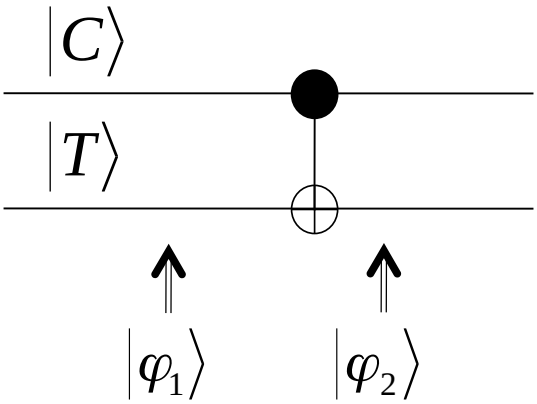
The Swap gate, shown in Figure 3.9, does exactly what it names implies: it swaps two qubits and can be represented by the following matrix, where the 1s in rows 2 and 3 trigger the *'swapping'* mathematically.

Swap gate matrix: $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Fredkin Gate

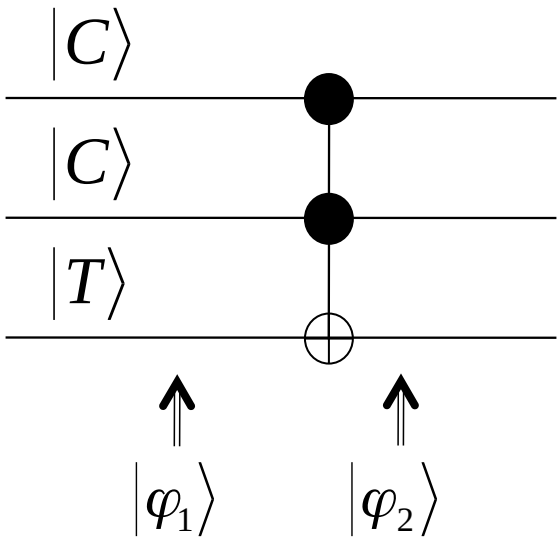
The Fredkin gate, shown in Figure 3.9, is nothing more than a controlled swap that takes 3-qubits, where one qubit is the control qubit, and the other two qubits are the target qubits, which may or may not be swapped depending on the control qubit. If the control qubit is $|0\rangle$, then there is no change, and the two target qubits are not swapped. On the other hand, if the control qubit is $|1\rangle$ then the target qubits will swap states.

Fredkin gate matrix: $\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$



C: *control bit*
T: *target bit*

Figure 3.6: CNOT quantum gate



C: *control bit*
T: *target bit*

Figure 3.7: Toffoli quantum circuit gate

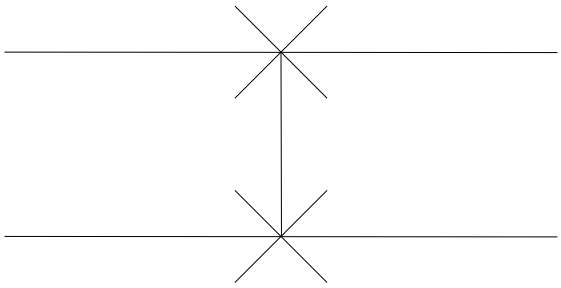


Figure 3.8: SWAP gate

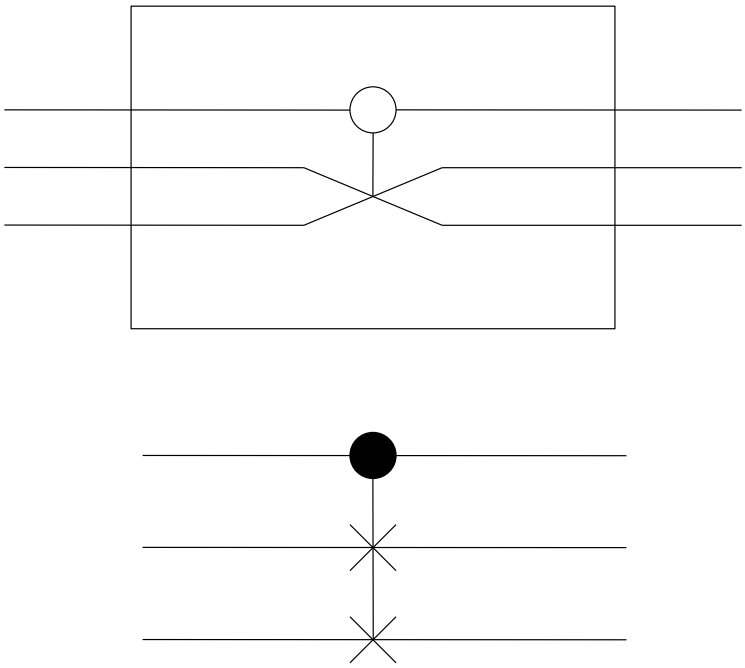


Figure 3.9: Fredkin quantum circuit gate

3.4 Entanglement and Teleportation

3.4.1 What is Entanglement, and How Does it Work?

Quantum entanglement is a property of quantum mechanics where two or more quantum particles, in this case qubits, are linked (entangled) together in a fundamental way; a change in the environment of one particle directly affects the other particle. The behavior of the two particles is linked, even if one particle were physically sent over a vast distance such as *'space.'* Suppose that the other particle, which is entangled with the one sent through *'space,'* stays put on Earth. When observing, or measuring, the particle that remained on Earth, the remote particle is in-

stantly affected. This interaction is immediate: it is faster than the speed of light. Entanglement is at the heart of a paper by Einstein et al. [24], often referred to as the EPR paradox.

How does this apply to qubits? First, we take two qubits and entangle them such that when one is measured, there is a 50% chance of measuring $|0\rangle$ and a 50% chance of measuring $|1\rangle$. However, once a measurement is performed on one-half of the entangled pair, the quantum system collapses, and there is a 100% chance that the other qubit will have the same value as the first when it is measured. Note that this does not make it possible for instantaneous communication across arbitrary distance. Entanglement is expressed mathematically as $\frac{|00\rangle+|11\rangle}{\sqrt{2}}$.

3.4.2 What is Teleportation and How Does it Work?

Quantum teleportation uses entanglement to send an arbitrary quantum state from one location to another (e.g., from Alice to Bob). Due to the no-cloning theorem, the quantum state at Alice's location is destroyed, but the state is recreated at Bob's location. A classical communication channel is required between Alice and Bob. Note that this is not quantum key distribution (teleportation could be used to distribute keys, but it would probably be rather cumbersome). Teleportation in turn can be used as a building block for quantum computers.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4:
Quantum Algorithms

Introduction

Richard Feynman suggested the notion of a quantum computer in 1982 [25]. His reasoning was that no classical computer could simulate quantum mechanical systems. On the other hand, a [quantum] computer, based on the laws of quantum mechanics, could simulate quantum mechanical processes more accurately and more efficiently than classical computers.

In the relatively short period of time since then, progress has been made towards this goal. Isaac Chuang et al. first pioneered the use of nuclear magnetic resonance (NMR) to build a quantum computer, which demonstrated Shor's factoring algorithm [10]. The NMR quantum computer used 7-qubits to factor the number 15 into its prime factors of 3 and 5.

As progress is made in physical implementation of quantum bits, the need for quantum algorithms will increase. Several algorithms have already been developed. Deutsch's Algorithm and the Deutsch-Jozsa Algorithm are simple algorithms that demonstate quantum parallelism. Grover's algorithm is an algorithm for searching unordered lists. Shor's algorithm, as mentioned above, is designed for factoring.

4.1 Quantum Parallelism

A quantum computer's ability to exploit superposition is called quantum parallelism and provides an advantage over classical computers. A classical computer would traditionally evaluate a function, $f(x)$, for one value of at time. In contrast, a quantum computer can evaluate $f(x)$ for multiple values of x simultaneously.

4.2 Deutsch's Algorithms

One of the earliest algorithms was designed by David Deutsch, a physicist from Oxford University. Deutsch's algorithm [26] exploits quantum parallelism. Quantum parallelism is a fundamental feature of Deutsch's algorithm that allows a quantum computer to evaluate a function, $f(x)$, for many different values of x at the same time. For a function $f(x)$ whose range is $\{0, 1\}$ and whose domain is $\{0, 1\}$, Deutsch's algorithm is able to determine whether $f(x)$ is either constant $f(0) = f(1)$ or balanced $f(0) \neq f(1)$ with only one evaluation of $f(x)$. On a classical

computer, $f(x)$ would have to be evaluated twice in order to determine whether it is constant or balanced. Deutsch's algorithm by itself is not very useful, but it demonstrates quantum parallelism. A variation on this algorithm is the Deutsch-Jozsa algorithm, in which $f(x)$ has a range of $\{0, 1\}$ and a domain of the natural numbers between 0 and $2^n - 1$. The function $f(x)$ is constant if half of the inputs go to 0 and half go to 1; $f(x)$ is balanced if all of the inputs go to 0 or all of them go to 1. More formally:

Suppose $f(x) : \{0, 1\} \rightarrow\{\ 1, 0\}$ is a function with a one-bit domain and range. Deutsch's Algorithm evaluates $f(x)$ for all values of x simultaneously. The figure below shows a 2-qubit quantum circuit that implements Deutsch's Algorithm. The initial state of the quantum circuit is $|x, y\rangle$= $|0, 1\rangle$= $|0\rangle\,|1\rangle$. We are interested in determining whether $f(0) \neq f(1)$ (f is balanced), or $f(0) = f(1)$ (f is constant).

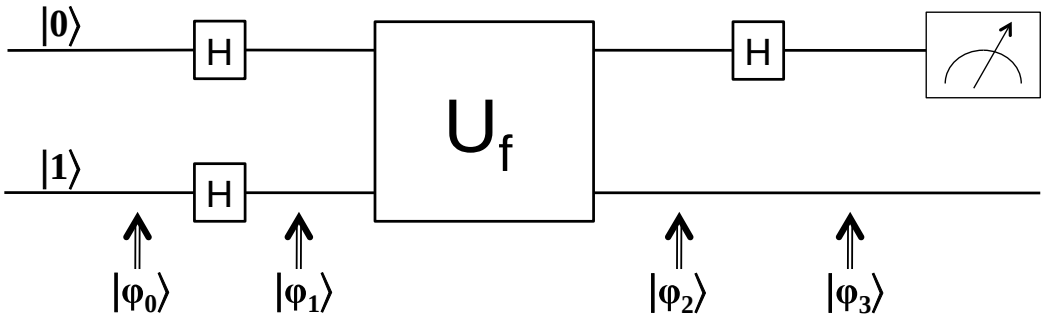


Figure 4.1: Quantum circuit for Deutsch's Algorithm

Figure 4.1 shows the quantum circuit for Deutsch's Algorithm, where U_f is a unitary operator. The details of U_f are not important and therefore can be thought of as a "black box."

To interpret the output of the algorithm, consider the stage of the algorithm when the quantum system has progressed to the state $|\varphi_3\rangle$.

$$|\psi_3\rangle = \begin{cases} \pm|0\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right] \text{ if } f(0) = f(1) \\ \\ \pm|1\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right] \text{ if } f(0) \neq f(1) \end{cases} \tag{4.1}$$

$$|\varphi_3\rangle = \pm\,|f(0) \oplus f(1)\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right], \tag{4.2}$$

where $|f(0) \oplus f(1)\rangle$ is 0 if $f(0) = f(1)$ or 1 if $f(0) \neq f(1)$.

4.3 The Deutsch-Jozsa Algorithm

A more sophisticated version of Deutsch's Algorithm is the Deutsch-Jozsa Algorithm. This algorithm provides more speedup than Deutsch's Algorithm by evaluating $f(x)$ on a register of n quantum bits in a superposition of all values between 0 and $2^n - 1$. Deutsch's Algorithm is a special case of the Deutsch-Jozsa Algorithm, in which $n = 1$. The premise for this algorithm is the same as Deutsch's Algorithm, with a subtle difference. More formally,

Given $f(x) : \{0,1\}^n \rightarrow \{1,0\}$, with the same range and the function$f(x)$ used in Deutsch's Algorithm, evaluate $f(x)$ in a single iteration and determine whether $f(x)$ is constant or balanced. The function $f(x)$ is constant if the evaluation of $f(x)$ is the same on all inputs; $f(x)$ is balanced if it is equal to 1 for half the inputs and 0 for other half.

See Figure 4.2 for the quantum circuit, with the initial state of the quantum circuit as

$$|x, y\rangle = |0\rangle^{\otimes n} |1\rangle \tag{4.3}$$

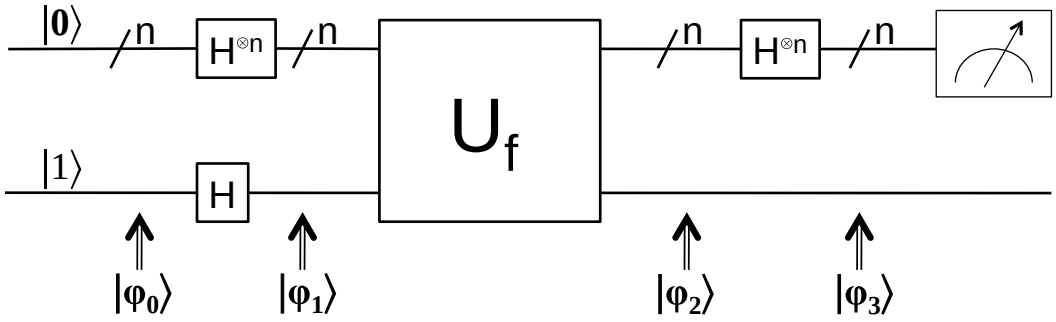


Figure 4.2: Quantum circuit for the Deutsch-Jozsa Algorithm

4.4 Grover's Algorithm

Lov Grover showed in [13] that unlike classical algorithms that can search an unsorted database in linear time, it is possible to search an unordered list with a quantum search algorithm in $O(n^{\frac{1}{2}})$ time. The quantum circuit for Grover's Algorithm is shown in Figure 4.3.

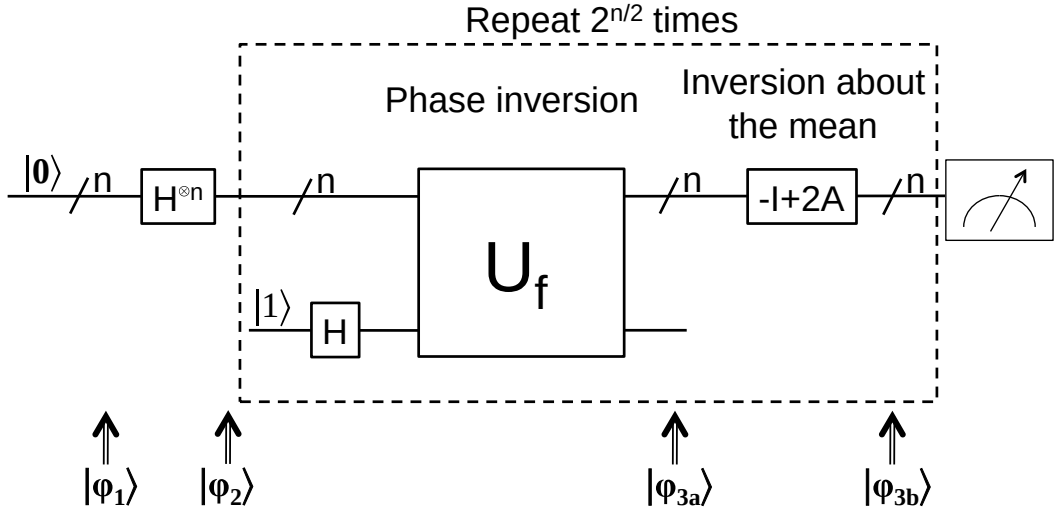


Figure 4.3: Quantum circuit for Grover's Algorithm

4.5 Shor's Algorithm

Peter Shor first proposed an algorithm to factor numbers [9]. The power of this algorithm is that a quantum computer would be able to perform prime factorization of large numbers in polynomial time, on the order of $O\left(\log\ n\right)^3$, where n is the number of bits of the number to be factored. Shor's algorithm could factor a number exponentially faster than the best-known classical algorithm. To date the fastest known algorithm can only factor in time $\exp\left(\Theta\left(n^{\frac{1}{3}}\log^{\frac{2}{3}} n\right)\right)$.

Shor's algorithm is based on the quantum Fourier transform (QFT), which is an analogue to the discrete Fourier transform (DFT).

For more details on the DFT and QFT, see [22]. The quantum circuit for Shor's Algorithm is shown in figure 4.4.

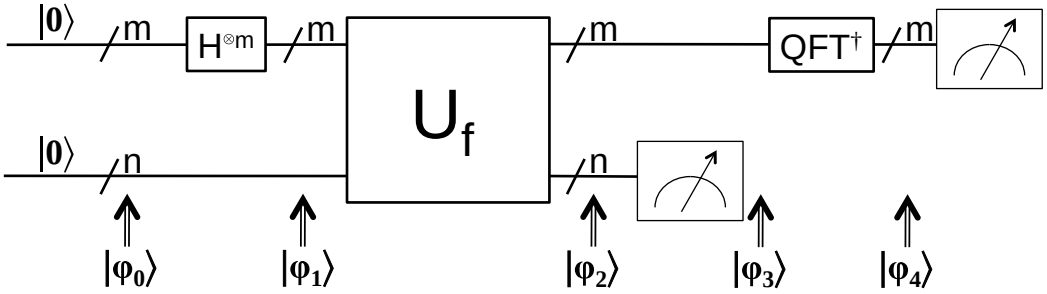


Figure 4.4: Quantum circuit for Shor's Algorithm

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 5:
Quantum Error Correction

Quantum error correction is used to protect the information that quantum bits contain. Errors are generally caused by decoherence and other noise from the environment. Quantum error correction schemes are essential for large-scale, fault-tolerant quantum computation. Quantum error correction must operate within the constraints of the no-cloning theorem, which forbids the copying of a quantum state.

Quantum error correction has a high cost because many physical quantum bits are required to implement a single logical quantum bit. Furthermore, a logical qubit may be represented by several logical qubits, which in turn are represented by several physical qubits (two levels of recursion in this example). There are many different kinds of error correction codes, and each code has adjustable parameters. The specific quantum error correcting code and parameters to be used depend on the reliability of the physical technology used to implement the quantum bits, the size of the algorithm, the algorithm being implemented, and other factors. Designing a large-scale, fault-tolerant quantum computer is a balancing act. A winning design must successfully balance the cost of error correction against the gains of quantum parallelism.

The following sections discuss three different types of quantum bit errors: a single bit-flip error, a single phase-flip error, and a single bit-flip or phase-flip error.

5.1 Bit Flip Errors

5.1.1 Classical Approach

Bit flip errors are common and the only kind of bit errors that occur with classical computers. Repetition codes provide a simple example of how classical computers solve bit flip errors. The idea is straightforward; a single bit, either a 0 or 1, needs to be transmitted from one point to another. To ensure the bit arrives in the state in which it left, one could simply repeat the bit, resulting in a string of bits. The receiver compares the individual bits with each other in the string. Based on a majority rule, the receiver can determine which bits were flipped. By simply flipping the bits back to their original state, it is possible to determine what should have been transmitted.

Example: 3-bit repetition code

$0 \mapsto 000$

$1 \mapsto 111$

A sender who wants to transmit the bit 0 encodes it using a 3-bit repetition code. The receiver reads 010. Based on the 3-bit repetition code, the receiver will know that the 1 is an error and simply flip the bit to read 000. This results in receiving the original string of bits and correctly interpreting that a 0 was transmitted.

5.1.2 Quantum Approach

The approach for correcting a quantum bit-flip error is not the same as the classical approach. The no-cloning theorem forbids observing the state of the system, since measurement collapses the quantum state. To overcome this, it is necessary to be clever to somehow indirectly learn about the error. To avoid collapsing the quantum state, quantum error correction uses quantum entanglement.

A three-bit repetition code can address a single bit-flip error. This quantum circuit uses entanglement to encode the data qubit.

Example:

The following example is from [22]. Suppose the initial state of $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$. The three-bit repetition code encodes $|\psi\rangle$ as $\alpha\,|000\rangle + \beta\,|111\rangle$ using three qubits:

$$|0\rangle \mapsto |\,0_L\rangle \equiv |000\rangle$$

$$|1\rangle \mapsto |\,1_L\rangle \equiv |111\rangle \tag{5.1}$$

The notation *L* means the encoded qubits $|000\rangle$ and $|111\rangle$ are from the logical $|0\rangle$ and $|1\rangle$ states, and not to be interpreted as the physical qubit states $|0\rangle$ and $|1\rangle$.

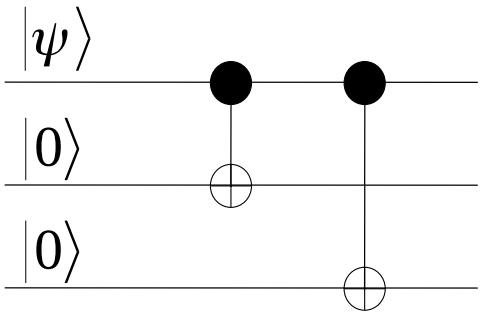


Figure 5.1: Encoding circuit for 3-qubit error correcting code for single bit-flip

The circuit in Figure 5.1 encodes the qubit using two cNOT gates. After encoding, a single bit-flip error may occur to any of the quantum bits due to noise in the environment. After this occurs, additional gates (not shown) detect and correct the error. Specifically, two cNOT gates, symmetric to the two cNOT gates shown, carry out a decoding process, which causes the two lower qubits (the syndrome qubits) to both become $|1\rangle$ if a bit-flip error occurred on the upper (data) qubit. To correct this error, a Toffoli gate (not shown), uses the two syndrome qubits as control bits to flip the data qubit if both syndrome qubits are $|1\rangle$.

After the qubit has been decoded (but before the Toffoli gate) there are four possible error cases.

1. No Error
   Result:

$$\alpha\,|000\rangle + \beta\,|111\rangle \mapsto \alpha\,|000\rangle + \beta\,|100\rangle \tag{5.2}$$

2. Bit flip on the first qubit
   Result:

$$\alpha\,|000\rangle + \beta\,|111\rangle \mapsto \alpha\,|111\rangle + \beta\,|011\rangle \tag{5.3}$$

3. Bit flip on the second qubit

Result:

$$\alpha\,|000\rangle + \beta\,|111\rangle \mapsto \alpha\,|010\rangle + \beta\,|110\rangle \tag{5.4}$$

4. Bit flip on the third qubit
   Result:

$$\alpha\,|000\rangle + \beta\,|111\rangle \mapsto \alpha\,|001\rangle + \beta\,|101\rangle \tag{5.5}$$

Based on these four cases, it is possible to determine where the error occurred so that the appropriate bit can be flipped to correct the error.

5.2 Phase Flip Errors

Another problematic error for quantum states is phase errors, commonly known as sign errors. In other words, a qubit undergoes the following transformation: $\alpha\,|0\rangle + \beta\,|1\rangle \mapsto \alpha\,|0\rangle - \beta\,|1\rangle$. This kind of error is different from bit flip errors because unlike the bit flip code, there is no classical counterpart to model a phase flip error.

With bit flips, the basis states were $|0\rangle$ and $|1\rangle$. For phase flips, the basis states will be $|+\rangle$ and $|-\rangle$, respectively. The encoding of the states resembles that of the three-qubit code for correcting bit-flip errors, resulting in the encoding below:

$$\begin{aligned} |+\rangle &\mapsto |0_L\rangle \equiv |+++\rangle \\ |-\rangle &\mapsto |1_L\rangle \equiv |---\rangle \end{aligned} \tag{5.6}$$

The encoding and decoding process will be the same as the bit flip code with one exception.

In order to compensate for the change in basis states, we apply the Hadamard gate and then its inverse, as shown in Figure 5.2. The inverse gate for a Hadamard gate is the Hadamard gate itself ($H \otimes H = I$). With the exception of applying the Hadamard gates, the three-qubit code for correcting a single phase-flip error is essentially the same as the three-qubit code for correcting a single bit-flip error and follows the same logic for correcting the errors.

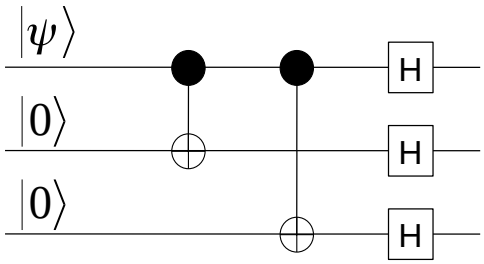


Figure 5.2: Encoding circuit for 3-qubit error correcting code for single phase-flip

5.3 Single Bit Flip or Phase Flip Correction

A quantum system could have both bit and phase flip errors. Shor developed a nine-qubit error correcting code in [27]. The nine-qubit code is a combination of the three-qubit bit and phase flip codes. It takes one qubit and encodes it using nine qubits. More formally:

- Logical qubit $|0\rangle$ is encoded as follows:

$$|0\rangle \mapsto \tfrac{1}{2\sqrt{2}} \left(|000\rangle + |111\rangle\right) \left(|000\rangle + |111\rangle\right) \left(|000\rangle + |111\rangle\right), \tag{5.7}$$

which is equivalent to

$$\begin{aligned} |0\rangle \mapsto \tfrac{1}{2\sqrt{2}} &(|000000000\rangle + |000000111\rangle + |000111000\rangle + |000111111\rangle \\ &+ |111000000\rangle + |111000111\rangle + |111111000\rangle + |111111111\rangle) \\ = \tfrac{1}{\sqrt{2}} &\left(|0_1 0_2 0_3\rangle + |1_1 1_2 1_3\rangle\right) \tfrac{1}{\sqrt{2}} \left(|0_4 0_5 0_6\rangle + |1_4 1_5 1_6\rangle\right) \tfrac{1}{\sqrt{2}} \left(|0_7 0_8 0_9\rangle + |1_7 1_8 1_9\rangle\right) \end{aligned} \tag{5.8}$$

- Logical qubit $|1\rangle$ is encoded as follows:

$$|1\rangle \mapsto \tfrac{1}{2\sqrt{2}} \left(|000\rangle - |111\rangle\right) \left(|000\rangle - |111\rangle\right) \left(|000\rangle - |111\rangle\right), \tag{5.9}$$

which is equivalent to

$$\begin{aligned} |1\rangle \mapsto & \tfrac{1}{2\sqrt{2}} (|000000000\rangle - |000000111\rangle - |000111000\rangle + |000111111\rangle \\ & - |111000000\rangle + |111000111\rangle + |111111000\rangle - |111111111\rangle) \\ = & \tfrac{1}{\sqrt{2}} \left(|0_1 0_2 0_3\rangle - |1_1 1_2 1_3\rangle\right) \tfrac{1}{\sqrt{2}} \left(|0_4 0_5 0_6\rangle - |1_4 1_5 1_6\rangle\right) \tfrac{1}{\sqrt{2}} \left(|0_7 0_8 0_9\rangle - |1_7 1_8 1_9\rangle\right) \end{aligned} \tag{5.10}$$

Where the first, fourth and seventh qubit of each encoding accounts for phase flips, and each set of states (1,2,3), (4,5,6) and (7,8,9) accounts for bit flips [28]. Figure 5.3 shows the encoding circuit used with Shor's nine-qubit error correction code.

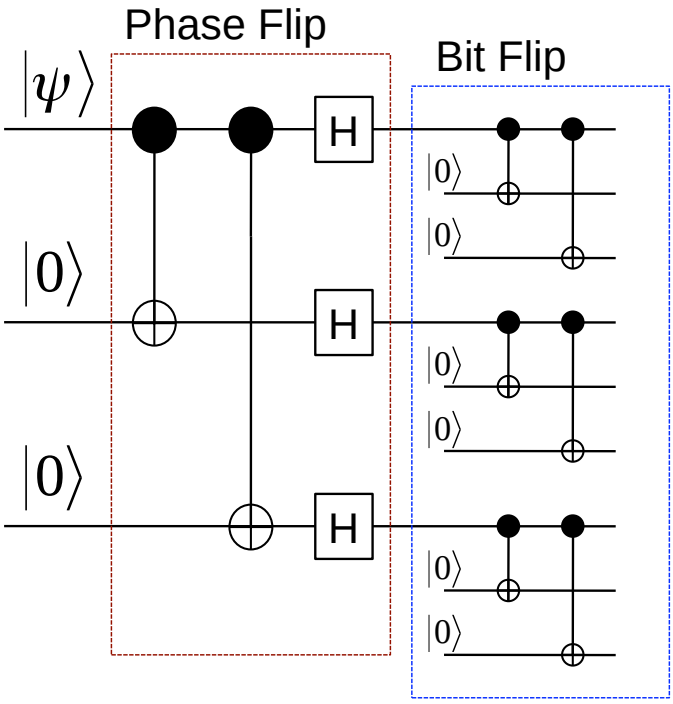


Figure 5.3: Encoding circuit for Shor's 9-qubit error correcting code

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 6:
Physical Implementations

There are a number of competing technologies for the physical implementation of quantum bits. In order for a technology to be useful, it must meet a basic set of criteria first proposed by David DiVincenzo in [29], in which he discusses seven requirements, the last two focusing on quantum communication. The five requirements for the physical implementation of a quantum computer put fourth by DiVincenzo are as follows:

1. A scalable physical system with well-characterized qubits.
2. The ability to initialize the state of the qubits to a simple fiducial state, such as $|000\ldots\rangle$.
3. Long relevant decoherence times, much longer than the gate operation time ($> 10^4$ operation time).
4. A universal set of quantum gates.
5. A qubit-specific measurement capability.

These requirements are fundamental, but achieving them is an enormous engineering challenge. As a result, no technology currently exists that satisfies all of the requirements sufficiently well to reach the threshold of reliability required for the realization of large-scale, fault-tolerant quantum computers. While there is no clear favorite, the following sections will discuss several prominent technologies and their advantages and disadvantages.

6.1 Technologies

6.1.1 Nuclear Magnetic Resonance

Nuclear magnetic resonance (NMR) spectroscopy is a technique used in chemistry to measure the properties of liquids, solids and gases in order to determine the structure of the molecules being observed. NMR uses the nuclear spins of a system of molecules by applying a radio-frequency electromagnetic pulse. The objective is to manipulate and detect the nuclear spins of the molecules in order to create a resonance frequency that can be measured. The spins of the molecules are what become the qubits and essentially the quantum computer.

NMR has two basic parts, the spectrometer and a liquid sample of molecules. The liquid sample is placed within a magnetic field, and the spectrometer applies an electromagnetic pulse to the sample. This in turn manipulates the nuclear spin states of the molecules within the liquid sample. This description of how an NMR quantum computer works is very basic, but allows the reader to develop a general image of the system. The first experimental realization of NMR implemented an order-finding algorithm [30] followed shortly by an NMR machine that implemented Shor's algorithm [10].

Challenges

NMR faces scalability issues because each additional quantum bit requires an additional atom in the molecule, resulting in larger, more cumbersome molecules. Warren et al. address the usefulness of NMR quantum computing machines [31], and they predict that NMR is not viable as a technology for large-scale quantum computers.

6.1.2 Superconductivity

The idea of superconductivity is not new, but researchers have proposed the use of superconductivity as a technology for the physical implementation of quantum computers.

> [Superconductivity has shown researchers] promising approaches to quantum computing because it offers devices with little dissipation, ultra-sensitive magnetometers, and electrometers for state readout, large-scale-integration, and a family of classical electronics that could be used for quantum bit control. [2]

In other words, the use of superconductors allows for better control of quantum bits. Superconductivity aides in the ability to control quantum bits, entangle the quantum bits, and increase decoherence times in the face of environmental disturbances. Superconductors are flexible in the fact that several different kinds of qubits can be implemented within the same superconductive circuit through the use of Josephson Junctions. Josephson junctions are circuits based on the Josephson effect [32], in which a weak current flows through two superconductors separated by a thin, non-conducting, insulating barrier. This effect allows for Josephson junctions to easily manipulate the qubits tunneling through the barrier, resulting in control of the qubit. For a detailed explanation of the different types of qubits and implementation of Josephson junctions, see [33, 34]

Berggren [2] supports the use of superconductive quantum computers based on the following key features:

1. It is a coherent quantum technology
2. Superconductive devices can be fabricated and integrated monolithically
3. Nearly ideal superconductive sensors exist
4. A superconductive classical electronics family exists for device control

Challenges

A Potential drawback to using superconductivity is that the operational temperatures are near absolute zero. Extreme cold is imperative to lessen the effects of decoherence times. Not only must temperatures be cold enough to allow for superconductivity to take place, but the device must also be isolated from environmental disturbances.

6.1.3 Quantum Dots

Quantum dots are semiconductors known as nanocrystals found in the electronics of personal computers or in light emitting diodes (LEDs). The properties of quantum dots make them attractive as a physical implementation technology because a high level of control of the nanocrystals can be achieved. The ability to control the flow of electrons through a quantum dot also facilitates precise measurement. Imamoğlu [35] exploits the fact that electrons within quantum dots have longer decoherence times. Quantum dots meet the requirements for longer decoherence times as stated by DiVincenzo in [29]. Imamoğlu [35] proposed the use of quantum dots because quantum dots could provide scalability to over 100 qubits, long decoherence times, and fast, long-distance interactions between qubits with the use of a micro-cavity. See [36, 37] for further explanation of the implementation of quantum dots for quantum computation and information processing.

6.1.4 Linear Optics

Quantum computers that utilize linear optics show promise because many of the optical components are off-the-shelf, and experimental demonstrations of logic gates and memory devices have been shown to work [38]. Linear optics based quantum computers will use photons to implement quantum bits and gates. The quantum bits will use the polarization of the photon in order to encode a 0 or 1.

Challenges

See [39] for other sources documenting the strengths, weaknesses, and challenges of realizing linear optics based quantum computing. In summary, some of the challenges are:

- Inability to store photons.
- Low photon creation rates
- Low detection efficiencies.
- Scalability is limited due to detection inefficiencies.

6.1.5 Ion Traps

An ion trap is a device that uses electromagnetic fields to manipulate ions within the field. A device used as an ion trap for quantum computing is the Paul Trap, also known as a quadrupole ion trap invented by [40]. [41, 42] were the first to propose and implement a CNOT gate, respectively, by using trapped ions in order to perform quantum computation. Many researchers are actively pursuing ion traps because they have longer decoherence times, and they satisfy the requirements of DiVincenzo [29]

Challenges

Even though researchers have been able to put an ion trap onto a semiconductor, it still remains a challenge. The challenge is the scaling of the ion traps in order to incorporate enough qubits to become useful.

6.2 Decoherence Times

All the different technologies referenced above have to address decoherence times. Some technologies are more susceptible to shorter decoherence times than others, but the decoherence time will need to be more than 10^4 times greater than the operation time [29].

6.3 Scalability

Scalability is a major hurdle that competing technologies must overcome to realize large-scale, fault-tolerant quantum computers . For example, superconductivity shows more promise for scalability than other approaches that cannot exploit conventional microchip manufacturing methods [2]. Figure 6.2 shows the point at which a quantum computer outperforms a classical computer for factoring [2]. [2] cites [43] that a 2048-bit number would take more time to factor than the age of the universe on a classical computer. However, a quantum computer with

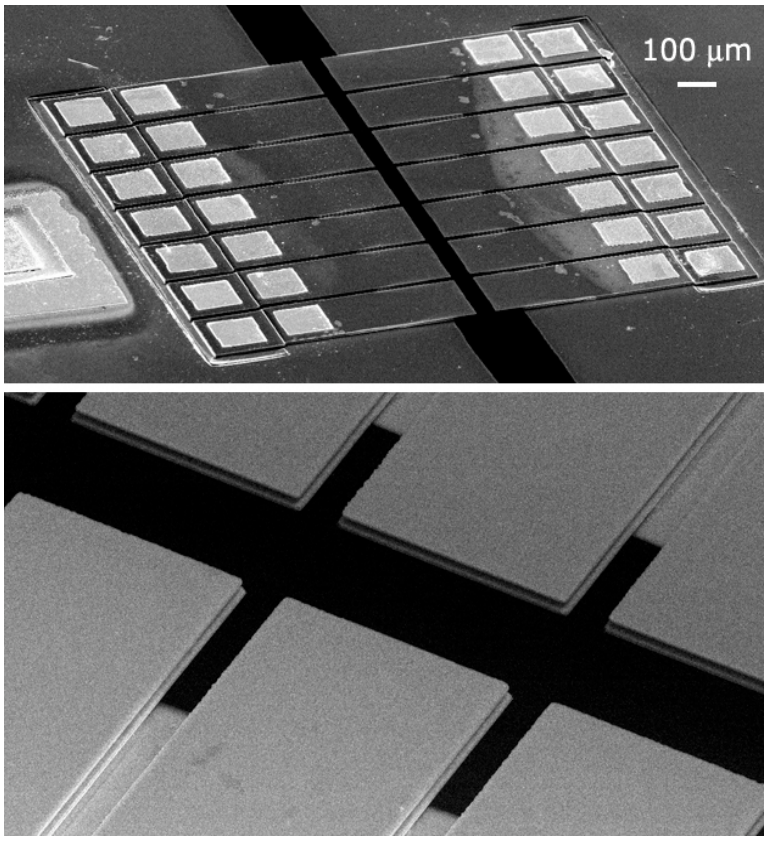


Figure 6.1: TOP: Ion trap chip. BOTTOM: Magnified view. Figure taken from [1].

a clock speed of 100 MHz, that is far slower than today's personal classical computer, would be able to factor a 2048-bit number in one hour. In order to produce a quantum computer capable of this, estimates stated in [2] taken from [43, 44] suggest a quantum computer would need approximately 1 million qubits regardless of the technology being used to implement a quantum computer.

Tzvetan Metodi, in his book [14], summarizes scalability requirements in order to build a quantum computer, and they are:

- Reliable and realistic implementation technology, that adheres to the DiVincenzo requirements [29] for implementing quantum computation.
- Robust, fault-tolerant structures encoded using efficient error correction algorithms. This requirement provides system-level fault tolerance that will allow the execution of an arbitrarily large sequence of universal quantum logic operations within the architecture decoherence time.

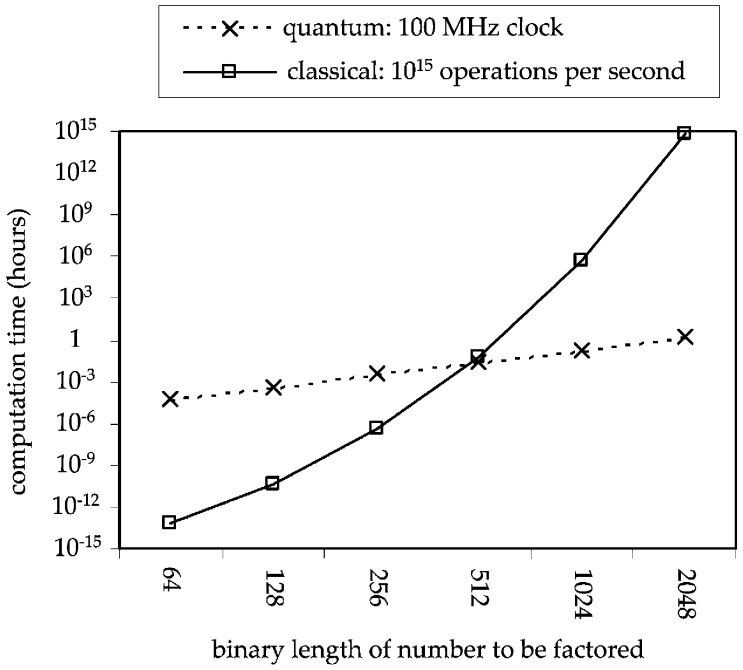


Figure 6.2: Comparison of operation time between a classical and quantum computer. Objective was to factor a 2048-bit number. Figure taken from [2]

- Efficient quantum resource distribution at both the application level and the physical qubit level that allows maximum overlap of computation and error correction.

CHAPTER 7:
Quantum Circuit Workflow

7.1 Quantum Circuit Design Tool Concept of Operation

The quantum circuit designer application is built using the following command prompt input:

```
% javac app.java
```

The quantum circuit designer application is invoked using the following command prompt input:

```
% java app [circuit_file]
```

The command-line parameter circuit_file is the name of the file containing a description of the quantum circuit to be edited. To design a new quantum circuit from scratch, do not specify circuit_file.

As shown in Figure 7.1, the quantum circuit is displayed in the scrollable panel in the upper portion of the application window. The text description of the circuit is displayed in the scrollable text area located in the lower right portion of the application window. A variety of controls for specifying individual circuit elements is located in the lower left portion of the application window. These controls are:

- Hadamard Gate (H)
- Controlled NOT Gate (CNOT). Specifies a CNOT gate in which the control and target quantum bits are adjacent.
- Measurement Gate (M)
- Control Bit of CNOT Gate (Control). Specifies the control bit of a CNOT gate in which the control and target quantum bits are not necessarily adjacent.
- Target Bit of CNOT Gate (Target). Specifies the target bit of a CNOT gate in which the control and target quantum bits are not necessarily adjacent.

- Unitary Transform (U). Specifies a unitary transform for quantum algorithms such as Deutsch's Algorithm, The Deutsch-Jozsa Algorithm, and Simon's Periodicity Algorithm. This gate spans all the bits of the quantum circuit.
- Upside-Down Toffoli Gate Target Bit (UT1). Specifies the target bit of an upside-down Toffoli gate.
- Upside-Down Toffoli Gate Upper Control Bit (UT2). Specifies the upper control bit of an upside-down Toffoli gate.
- Upside-Down Toffoli Gate Lower Control Bit (UT3). Specifies the lower control bit of an upside-down Toffoli gate.
- Number of Quantum Bits (Pull-Down Menu)
- Toffoli Gate Upper Control Bit (T1). Specifies the upper control bit of a Toffoli gate.
- Toffoli Gate Lower Control Bit (T2). Specifies the lower control bit of a Toffoli gate.
- Toffoli Gate Target Bit (T3). Specifies the target bit of a Toffoli gate.
- Initial State of Quantum Bits. Enter the index of the quantum bit to be set in the first text field, enter the value (zero or one) in the second text field, then press the Set button.
- Pauli-X Gate (X)
- Pauli-Y Gate (Y)
- Pauli-Z Gate (Z)
- Unitary Transform Text Area. Specifies the matrix for the unitary transform U. After entering the values of the matrix (space-delimited columns and newline-delimited rows), press the Set U button.
- Phase Gate (S)
- $\pi/8$ Gate (T)
- Upper Bit of Swap Gate (S1). Specifies the upper bit of a swap gate.
- Lower Bit of Swap Gate (S2). Specifies the lower bit of a swap gate.
- Controlled-Phase Gate Control Bit (CS1). Specifies the control bit of a controlled-phase gate.
- Controlled-Phase Gate Target Bit (CS2). Specifies the target bit of a controlled-phase gate.
- Upside-Down Controlled-Phase Gate Target Bit (UCS1). Specifies the target bit of an upside-down controlled-phase gate.

- Upside-Down Controlled-Phase Gate Control Bit (UCS2). Specifies the control bit of an upside-down controlled-phase gate.
- Controlled-$\pi/8$ Gate Control Bit (CT1). Specifies the control bit of a controlled-$\pi/8$ gate.
- Controlled-$\pi/8$ Gate Target Bit (CT2). Specifies the target bit of a controlled-$\pi/8$ gate.
- Upside-Down Controlled-$\pi/8$ Gate Target Bit (UCT1). Specifies the target bit of an upside-down controlled-$\pi/8$ gate.
- Upside-Down Controlled-$\pi/8$ Gate Control Bit (UCT2). Specifies the control bit of an upside-down controlled-$\pi/8$ gate.
- Unitary Transform Text Area. Specifies the matrix for an additional unitary transform V (Inversion About the Mean) used in Grover's Algorithm. This gate does not necessarily span all the bits of the quantum circuit. After entering the values of the matrix (space-delimited columns and newline-delimited rows), press the Set V button.
- Controlled-Phase Gate Control Bit (CS1). Specifies the control bit of a controlled-phase gate.
- Controlled-Phase Gate Target Bit (CS2). Specifies the target bit of a controlled-phase gate.
- Upside-Down Controlled-Phase Gate Target Bit, Adjoint (UCSP1). Specifies the target bit of an upside-down controlled-phase gate (adjoint).
- Upside-Down Controlled-Phase Gate Control Bit, Adjoint (UCSP2). Specifies the control bit of an upside-down controlled-phase gate (adjoint).
- Controlled-$\pi/8$ Gate Control Bit, Adjoint (CTP1). Specifies the control bit of a controlled- $\pi/8$ gate (adjoint).
- Controlled-$\pi/8$ Gate Target Bit, Adjoint (CTP2). Specifies the target bit of a controlled-$\pi/8$ gate (adjoint).
- Upside-Down Controlled-$\pi/8$ Gate Target Bit, Adjoint (UCTP1). Specifies the target bit of an upside-down controlled-$\pi/8$ gate (adjoint).
- Upside-Down Controlled-$\pi/8$ Gate Control Bit (UCTP2). Specifies the control bit of an upside-down controlled-$\pi/8$ gate (adjoint).

To change the number of quantum bits in the circuit, use the pull-down menu. To insert an individual circuit element, click the button of the desired circuit element and then click on the

desired location in the scrollable panel in the upper portion of the application window. As the circuit is composed, a text description of the circuit is displayed in the scrollable text area in the lower right portion of the application window. This text can be copied and pasted to a file at any time.

7.2 Quantum Circuit Simulator Concept of Operation

The simulator is a separate program from the designer, and it should be placed in a separate folder from the simulator. The simulator reads in a text file to render the display of the quantum circuit, section 7.2.1 shows a detail circuit file and example. The simulator is compiled using the following command prompt input:

```
% javac app.java
```

The simulator is invoked using the following command prompt input:

```
% java [-ms2048m -mx2048m] app circuit_file
```

Figure 7.2 shows a screen shot of the simulator application. The simulator displays the quantum circuit in the scrollable panel in the upper portion of the application window. The red line indicates the current stage of the simulation. Since quantum circuits are read from left to right, the red line is initially on the left. The Next button advances the simulation by one step. The *Auto* button runs the simulation from start to finish without requiring any user interaction between intermediate stages of the simulation, and the simulator will output a file containing both the current state of the quantum system (a 2^N-by-1 column vector, where N is the number of quantum bits in the circuit) as well as the previous quantum operation (a 2^N-by-2^N matrix). The scrollable text area in the lower right hand portion of the application window displays both the current state of the quantum system as well as the previous quantum operation. The current state of the quantum system is also depicted visually in the lower left hand portion of the application window. The real amplitudes are displayed above the complex amplitudes. Positive real amplitudes are depicted as white bars, where the height of the bar corresponds to the amplitude, and negative real amplitudes are depicted in red. Positive complex amplitudes are depicted in grey, and negative complex amplitudes are depicted in pink.

7.2.1 Quantum Circuit File Format

The quantum circuit text description format is extremely simple, see figure 7.1 below. Consider the following example for Grover's Algorithm:

```
Define N 4 # Specifies the number of quantum bits
Define U 16 # Specifies the size of unitary matrix U
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 # First row of matrix U
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 # Second row of matrix U
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 # Third row of matrix U
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 # ...
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
Define V 8 # Specifies the size of unitary matrix V
-0.75 0.25 0.25 0.25 0.25 0.25 0.25 0.25 # First row of V
0.25 -0.75 0.25 0.25 0.25 0.25 0.25 0.25 # Second row of V
0.25 0.25 -0.75 0.25 0.25 0.25 0.25 0.25 # Third row of V
0.25 0.25 0.25 -0.75 0.25 0.25 0.25 0.25 # ...
0.25 0.25 0.25 0.25 -0.75 0.25 0.25 0.25
0.25 0.25 0.25 0.25 0.25 -0.75 0.25 0.25
0.25 0.25 0.25 0.25 0.25 0.25 -0.75 0.25
0.25 0.25 0.25 0.25 0.25 0.25 0.25 -0.75
Define Phi0 # Specifies the initial state of the qubits
0 0 0 1
Define Transform1 # Specifies the first transform
```

```
H H H I
Define Transform2 # Specifies the second transform
I I I H
Define Transform3 # Specifies the third transform
U
Define Transform4 # Specifies the fourth transform
V I
Define Transform5 # Specifies the fifth transform
U
Define Transform6 # Specifies the sixth transform
V I
Define Transform7 # Specifies the seventh transform
M M M I
```

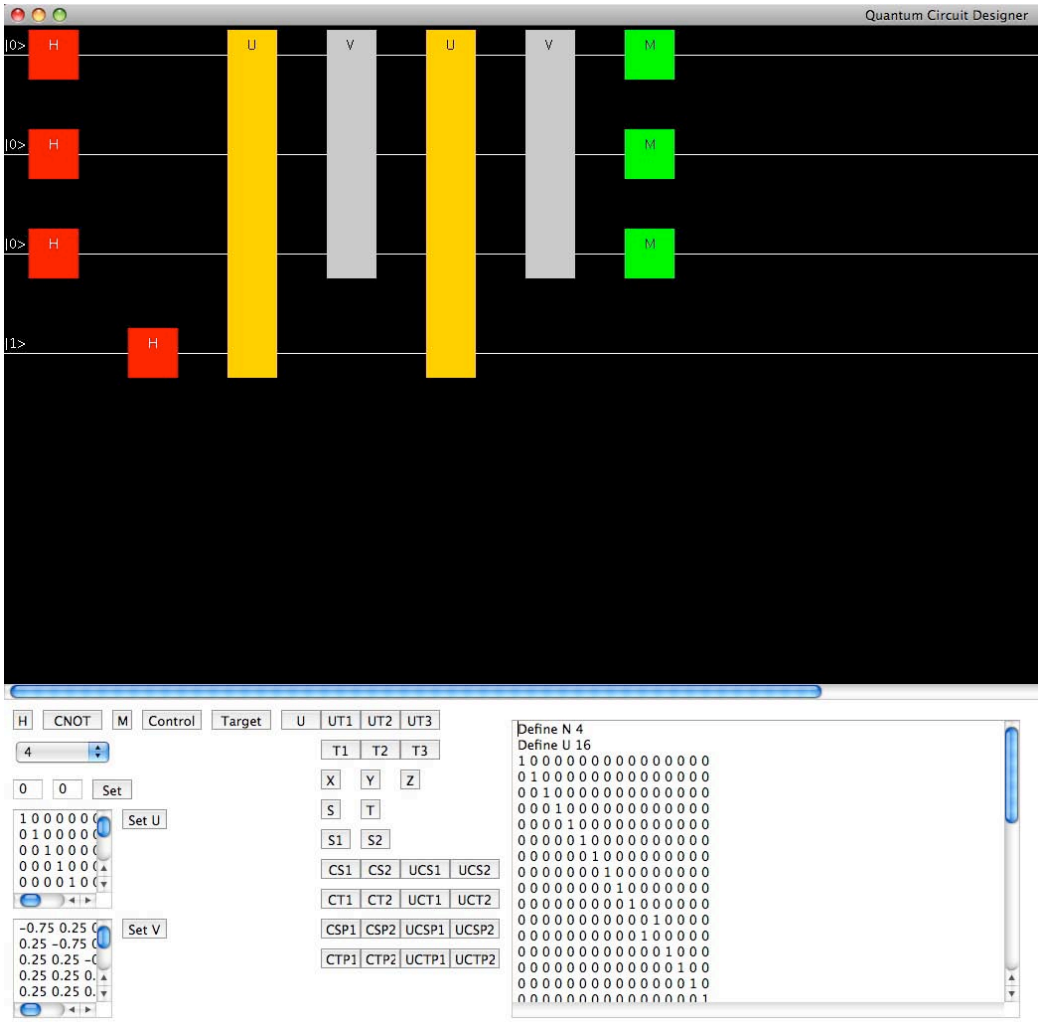


Figure 7.1: User interface of our quantum circuit designer

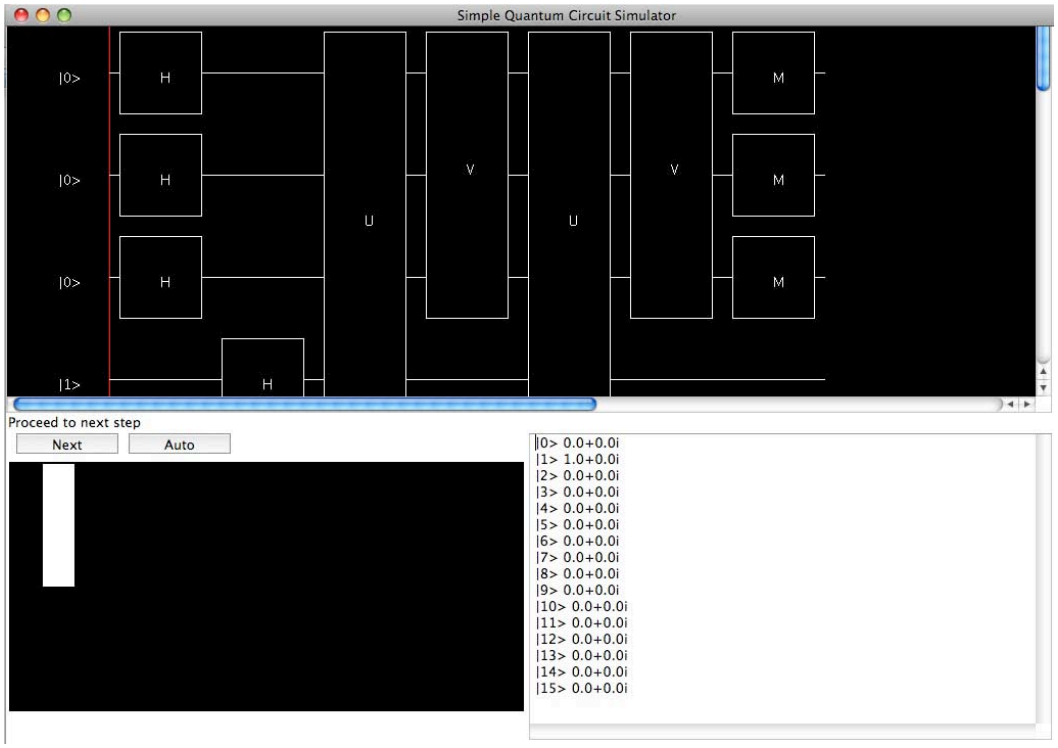


Figure 7.2: User interface of our quantum simulator

CHAPTER 8:
Demonstration of Workflow

8.1 Entanglement, Teleportation and Measurement

The following sections will demonstrate the use of the workflow to specify and simulate the quantum circuits for entanglement, teleportation, and measurement.

8.1.1 Measurement

If a quantum bit is in a superposition state, there is some probability that it will be a $|1\rangle$ upon measurement and some probability that it will be a $|0\rangle$ upon measurement. Measurement collapses the quantum state. Consider a quantum gate that uses Hadamard gates to put two quantum bits into a superposition; next, both quantum bits are measured:

```
Define N 2
Define Phi0
0 0
Define Transform1
H H
Define Transform2
M M
```

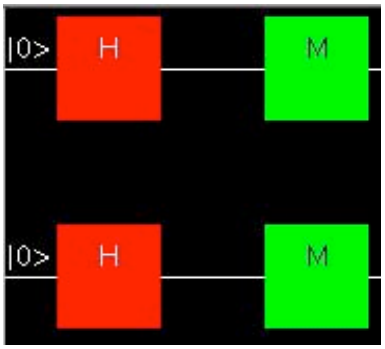


Figure 8.1: Measurement circuit example

The initial state of the two quantum bit system is $|00\rangle$:

$$
\begin{aligned}
|0\rangle &\mapsto 1.0 + 0.0i \\
|1\rangle &\mapsto 0.0 + 0.0i \\
|2\rangle &\mapsto 0.0 + 0.0i \\
|3\rangle &\mapsto 0.0 + 0.0i
\end{aligned}
\tag{8.1}
$$

After the H gates, the state is an equal superposition of $|00\rangle$, $|01\rangle$, $|10\rangle$, and $|11\rangle$:

$$
\begin{aligned}
|0\rangle &\mapsto 0.5000000000000007 + 0.0i \\
|1\rangle &\mapsto 0.5000000000000007 + 0.0i \\
|2\rangle &\mapsto 0.5000000000000007 + 0.0i \\
|3\rangle &\mapsto 0.5000000000000007 + 0.0i
\end{aligned}
\tag{8.2}
$$

The implementation of this simulator measures one quantum bit at a time. First the top quantum bit is measured, with the following message printed to standard out. In this run, the simulator randomly measured the top quantum bit as a $|1\rangle$ based on the probabilities (50% chance of $|0\rangle$ and 50% chance of $|1\rangle$). Now that it has been established that the first quantum bit is definitely a $|1\rangle$, the system is an equal superposition of $|10\rangle$ and $|11\rangle$ because the second quantum bit is still in an equal superposition of $|0\rangle$ and $|1\rangle$:

$$
\begin{aligned}
&\text{Qubit } 0 \text{ measured as } 1 \; |0\rangle \mapsto 0.0 + 0.0i \\
&|1\rangle \mapsto 0.0 + 0.0i \\
&|2\rangle \mapsto 0.7071067811865475 + 0.0i \\
&|3\rangle \mapsto 0.7071067811865475 + 0.0i
\end{aligned}
\tag{8.3}
$$

Next, the second quantum bit is measured in a similar fashion. In this run, it is measured as a $|0\rangle$. Since the first quantum bit is definitely a $|1\rangle$ and the second quantum bit is definitely a $|0\rangle$, the state of the two quantum bit system is $|10\rangle$, and the following message is printed to standard out:

Qubit 1 measured as 0 $|0\rangle \mapsto 0.0 + 0.0i$
$|1\rangle \mapsto 0.0 + 0.0i$
$|2\rangle \mapsto 1.0 + 0.0i$
$|3\rangle \mapsto 0.0 + 0.0i$

(8.4)

8.1.2 Entanglement

A simple entanglement circuit uses an H gate and a CNOT gate to entangle two qubits:

```
Define N 2
Define Phi0
0 0
Define Transform1
H I
Define Transform2
Control Target
```

The initial state of the two-qubit system is $|00\rangle$:

$$
\begin{aligned}
|0\rangle &\mapsto 1.0 + 0.0i \\
|1\rangle &\mapsto 0.0 + 0.0i \\
|2\rangle &\mapsto 0.0 + 0.0i \\
|3\rangle &\mapsto 0.0 + 0.0i
\end{aligned}
\tag{8.5}
$$

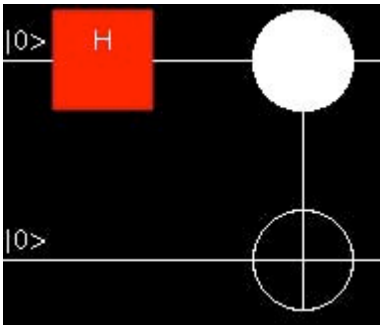


Figure 8.2: Entanglement circuit

After the H gate, the upper qubit is in an equal superposition of $|0\rangle$ and $|1\rangle$, and the lower qubit is still $|0\rangle$. Therefore, the two-qubit system is an equal superposition of $|00\rangle$ and $|10\rangle$:

$$
\begin{aligned}
|0\rangle &\mapsto 0.707106781186548 + 0.0i \\
|1\rangle &\mapsto 0.0 + 0.0i \\
|2\rangle &\mapsto 0.707106781186548 + 0.0i \\
|3\rangle &\mapsto 0.0 + 0.0i
\end{aligned}
\tag{8.6}
$$

After the CNOT gate, the two-qubit system is an equal superposition of $|00\rangle$ and $|11\rangle$. Therefore there is a 50% chance that one of the qubits will be a $|0\rangle$ or a $|1\rangle$, but once one qubit is measured

as a $|1\rangle$, the other one will also be a $|1\rangle$. Similarly, if one qubit is measured as a $|0\rangle$, the other one will also be a $|0\rangle$:

$$
\begin{aligned}
&|0\rangle \mapsto 0.707106781186548 + 0.0i \\
&|1\rangle \mapsto 0.0 + 0.0i \\
&|2\rangle \mapsto 0.0 + 0.0i \\
&|3\rangle \mapsto 0.707106781186548 + 0.0i
\end{aligned}
\tag{8.7}
$$

8.1.3 Teleportation

Teleportation is a method of reproducing a quantum state at another location, but it requires collapsing the quantum state at the original location. Suppose that Alice has a quantum state $|1\rangle$, and she wishes to reproduce this state at Bob's location. In the quantum circuit, the upper two qubits belong to Alice, and the lower qubit belongs to Bob. First, an H gate and a CNOT gate put the two lower qubits into a superposition. Thus, Alice and Bob have one half of an entangled pair of qubits. After two more quantum operations, Alice measures her two qubits and communicates the result of her measurement to Bob. There are four possibilities: Alice could measure $|00\rangle$, $|01\rangle$, $|10\rangle$, or $|11\rangle$. If Alice measures $|00\rangle$, Bob applies no correction to his qubit. However, if Alice measures $|01\rangle$, Bob applies an X gate to his qubit. If Alice measures $|10\rangle$, Bob applies a Z gate to his qubit. If Alice measures $|11\rangle$, Bob applies an X gate then a Z gate to his qubit.

```
Define N 3
Define Phi0
1 0 0
Define Transform1
I H I
Define Transform2
I CNOT
Define Transform3
CNOT I
Define Transform4
```

```
H I I
Define Transform5
M M I
```

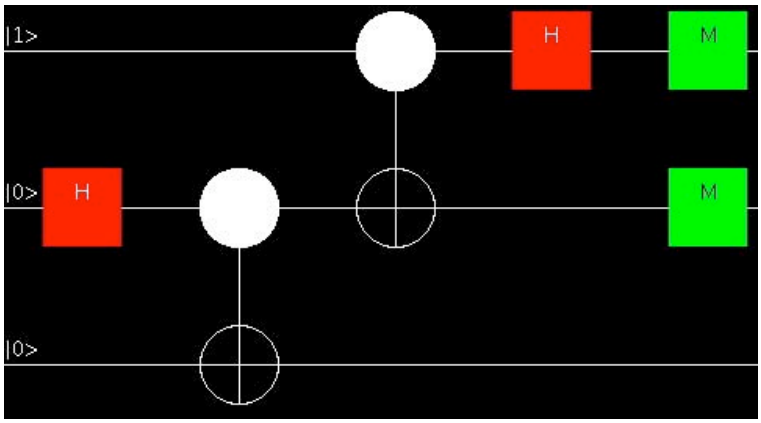


Figure 8.3: Teleportation circuit

In this run, Alice measures the upper qubit as $|1\rangle$, and she measures the middle qubit as $|0\rangle$:

$$
\begin{aligned}
|0\rangle &\mapsto 0.0 + 0.0i \\
|1\rangle &\mapsto 0.0 + 0.0i \\
|2\rangle &\mapsto 0.0 + 0.0i \\
|3\rangle &\mapsto 0.0 + 0.0i \\
|4\rangle &\mapsto 0.0 + 0.0i \\
|5\rangle &\mapsto -1.0 + 0.0i \\
|6\rangle &\mapsto 0.0 + 0.0i \\
|7\rangle &\mapsto 0.0 + 0.0i
\end{aligned}
\tag{8.8}
$$

Bob will apply the Z gate to correct the error so that his qubit is $|1\rangle$ rather than -$|1\rangle$.

8.2 Fourier Transforms

8.2.1 The Quantum Fourier Transform

The Quantum Fourier Transform (QFT) plays a key role in Shor's Algorithm. The QFT applies the following mathematical operation to the quantum state of the system (this example is the three-qubit QFT):

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \omega^4 & \omega^5 & \omega^6 & \omega^7 \\ 1 & \omega^2 & \omega^4 & \omega^6 & 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega & \omega^4 & \omega^7 & \omega^2 & \omega^5 \\ 1 & \omega^4 & 1 & \omega^4 & 1 & \omega^4 & 1 & \omega^4 \\ 1 & \omega^5 & \omega^2 & \omega^7 & \omega^4 & \omega & \omega^6 & \omega^3 \\ 1 & \omega^6 & \omega^4 & \omega^2 & 1 & \omega^6 & \omega^4 & \omega^2 \\ 1 & \omega^7 & \omega^6 & \omega^5 & \omega^4 & \omega^3 & \omega^2 & \omega \end{bmatrix}$$

In the above matrix, ω=$e^{i\pi/4}$. The following is the quantum circuit file:

```
Define N 3
Define Phi0
0 0 0
Define Transform1
H I I
Define Transform2
UCS1 UCS2 I
Define Transform3
UCT1 I UCT2
Define Transform4
I H I
Define Transform5
I UCS1 UCS2
Define Transform6
I I H
Define Transform7
S1 I S2
```

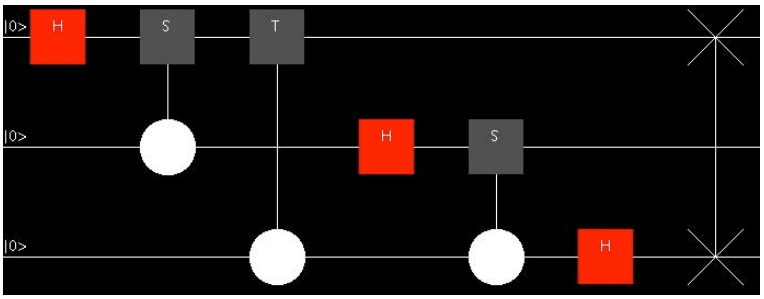


Figure 8.4: Circuit for the Quantum Fourier Transform

8.2.2 The Inverse Quantum Fourier Transform

The inverse QFT applies the inverse mathematical operation of the QFT. The quantum circuit is identical to the QFT, but it is run in reverse. In addition, the mathematical inverse of the quantum gates used in the QFT circuit are used in the inverse QFT circuit. The following is the quantum circuit file (this example is the three-qubit inverse QFT):

```
Define N 3
Define Phi0
0 0 0
Define Transform1
S1 I S2
Define Transform2
I I H
Define Transform3
I UCSP1 UCSP2
Define Transform4
I H I
Define Transform5
UCTP1 I UCTP2
Define Transform6
UCSP1 UCSP2 I
Define Transform7
H I I
```

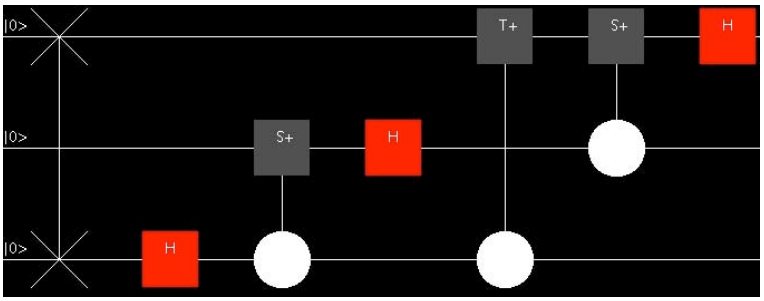


Figure 8.5: Circuit for the Inverse Quantum Fourier Transform

8.3 Quantum Algorithms

8.3.1 Deutsch's Algorithm

Deutsch's Algorithm is a simple quantum algorithm that demonstrates quantum parallelism. It determines whether a function $f(x)$ is constant or balanced. Both the domain and range of $f(x)$ are $\{0, 1\}$. A function is constant if all values of $f(x)$ are the same (i.e., $f(0) = f(1)$). A function is balanced if half the values of $f(x)$ are 1 and half are 0 (i.e., $f(0) \neq f(1)$). At the end of the algorithm, if the upper qubit is measured to be $|0\rangle$, $f(x)$ is constant, but if the upper qubit is measured to be $|1\rangle$, $f(x)$ is balanced. In this example, $f(0) = 1$, and $f(1) = 0$.

```
Define N 2 # This is a 2-qubit system
Define U 4 # Define the 4x4 matrix U
0 1 0 0
1 0 0 0
0 0 1 0
0 0 0 1
Define Phi0
0 1 # The initial state
Define Transform1
H H # Apply Hadamard gates to both qubits
Define Transform2
U # Apply 4x4 U gate to the 2-qubit system
Define Transform3
H I # Apply Hadamard gate to upper qubit
Define Transform4
```

```
M I # Measure upper qubit
```

After the last H gate is applied to the upper qubit, the two-qubit system is in an equal superposition of $|10\rangle$ and $|11\rangle$. Therefore, the top qubit is measured to be a $|1\rangle$, telling us that $f(x)$ is balanced. One evaluation of $f(x)$ (the matrix U) tells us whether $f(x)$ is constant or balanced. Classically, it would require two evaluations of $f(x)$ to determine this.

$$
\begin{aligned}
|0\rangle &\mapsto 0.0 + 0.0i \\
|1\rangle &\mapsto 0.0 + 0.0i \\
|2\rangle &\mapsto -0.7071067811865475 + 0.0i \\
|3\rangle &\mapsto 0.7071067811865475 + 0.0i
\end{aligned}
\tag{8.9}
$$

The following Java program generates the matrix U automatically:

```
class deutsch {
    public static final int SIZE=4;

    public static int f(int x)
    {
        if (x == 0)
            return 1;
        else if (x == 1)
            return 0;
    }

    public static void main(String args[])
    {
        int[][] matrix = new int[SIZE][SIZE];
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
```

```
                matrix[i][j] = 0;
            }
        }
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                int x = (j>>1)&1;
                int y = j&1;
                int xprime=x;
                int yprime=y^f(x);
                int row = (xprime<<1)|yprime;
                int col = j;
                matrix[row][col] = 1;
            }
        }
        System.out.println("Define U " + SIZE);
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                System.out.print("+matrix[i][j]);
                if (j != SIZE-1)
                    System.out.print(" ");
            }
            if (i != SIZE-1)
                System.out.println(" ");
        }
        System.out.println();
    }
}
```

8.3.2 The Deutsch-Jozsa Algorithm

The Deutsch-Jozsa Algorithm is similar to Deutsch's Algorithm, but the domain of $f(x)$ can be any natural number from 0 to $2^n - 1$, where n is the number of quantum bits. The upper qubits will be measured to be all 0 if $f(x)$ is constant. Any other measurement indicates that $f(x)$ is balanced. In this example, $f(0) = 1$, $f(1) = 1$, $f(2) = 0$, and $f(3) = 0$.

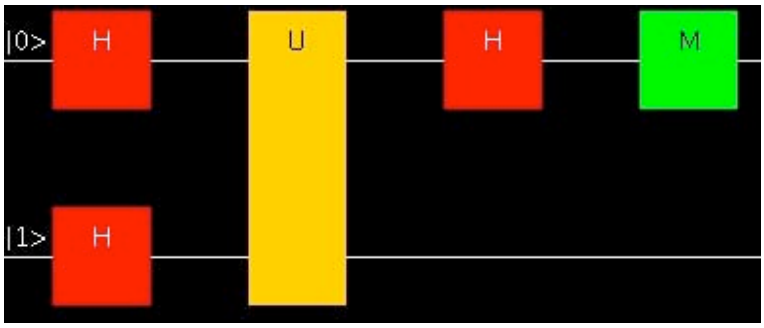


Figure 8.6: Circuit for Deutsch's Algorithm

```
Define N 3
Define U 8
0 1 0 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 1
Define Phi0
0 0 1
Define Transform1
H H H
Define Transform2
U
Define Transform3
H H I
Define Transform4
M M I
```

After the final H gates are applied to the upper and middle qubits, the three-qubit quantum system is in an equal superposition of $|100\rangle$ and $|101\rangle$. Therefore, the upper and middle qubits are measured to be $|10\rangle$. Since $|10\rangle$ is not $|00\rangle$, this indicates that the function is balanced.

$$
\begin{aligned}
&|0\rangle \mapsto 0.0 + 0.0i \\
&|1\rangle \mapsto 0.0 + 0.0i \\
&|2\rangle \mapsto 0.0 + 0.0i \\
&|3\rangle \mapsto 0.0 + 0.0i \\
&|4\rangle \mapsto -0.7071067811865498 + 0.0i \\
&|5\rangle \mapsto 0.7071067811865498 + 0.0i \\
&|6\rangle \mapsto 0.0 + 0.0i \\
&|7\rangle \mapsto 0.0 + 0.0i
\end{aligned}
\tag{8.10}
$$

The following Java program automatically generates the matrix U:

```
class dj {
    public static final int SIZE=8;

    public static int f(int x)
    {
        if (x == 0)
            return 1;
        else if (x == 1)
            return 1;
        else if (x == 2)
            return 0;
        else if (x == 3)
            return 0;
    }

    public static void main(String args[])
    {
        int[][] matrix = new int[SIZE][SIZE];
        for (int i = 0; i < SIZE; i++) {
```

```
            for (int j = 0; j < SIZE; j++) {
                matrix[i][j] = 0;
            }
        }
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                int x = (j>>1)&3;
                int y = j&1;
                int xprime=x;
                int yprime=y^f(x);
                int row = (xprime<<1)|yprime;
                int col = j;
                matrix[row][col] = 1;
            }
        }
        System.out.println("Define U " + SIZE);
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                System.out.print("+matrix[i][j]);
                if (j != SIZE-1)
                    System.out.print(" ");
            }
            if (i != SIZE-1)
                System.out.println(" ");
        }
        System.out.println();
    }
}
```

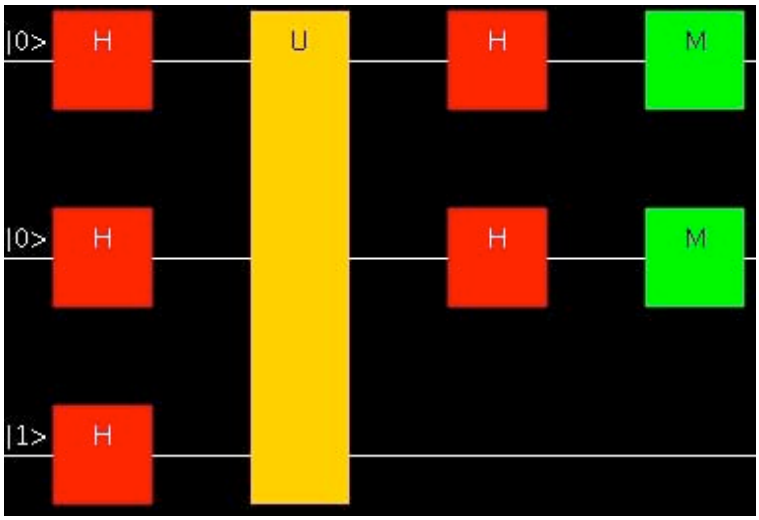


Figure 8.7: Circuit for the Deutsch-Jozsa Algorithm

8.3.3 Simon's Periodicity Algorithm

Simon's Algorithm finds patterns in functions. Suppose that the values of a function $f(x)$ repeat in some pattern. The goal of Simon's Periodicity Algorithm is to determine the period c of this pattern. The domain and range of function $f(x)$ can be any natural number between 0 and $2^n - 1$, where n is the number of quantum bits. It is necessary to run Simon's Periodicity Algorithm several times to determine c. Suppose

$f(0) = 4$,
$f(1) = 1$,
$f(2) = 5$,
$f(4) = 1$,
$f(5) = 4$,
$f(6) = 7$, and
$f(7) = 5$.

The period c is 5 because

$f(0) = f(0 \oplus c) = f(0 \oplus 5) = f(5)$,
$f(1) = f(1 \oplus c) = f(1 \oplus 5) = f(4)$,
$f(2) = f(2 \oplus c) = f(2 \oplus 5) = f(7)$,
$f(3) = f(3 \oplus c) = f(3 \oplus 5) = f(6)$,
$f(4) = f(4 \oplus c) = f(4 \oplus 5) = f(1)$,
$f(5) = f(5 \oplus c) = f(5 \oplus 5) = f(0)$,

$f(6) = f(6 \oplus c) = f(6 \oplus 5) = f(3)$, and
$f(7) = f(7 \oplus c) = f(7 \oplus 5) = f(2)$.

```
Define N 6 # This is a 6-qubit system
Define U 64 # Define the 64x64 matrix U
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0...
...
Define Phi0
0 0 0 0 0 0
Define Transform1
H H H I I I
Define Transform2
U
Define Transform3
H H H I I I
Define Transform4
M M M I I I
```

Prior to measurement of the top three qubits, the quantum state of the six-qubit system is an equal superposition of
$|000001\rangle$,
$|000010\rangle$,
$|000101\rangle$,
$|000111\rangle$,
$|010001\rangle$,
$|010100\rangle$,
$|010101\rangle$,
$|010111\rangle$,
$|101010\rangle$,
$|101100\rangle$,
$|101101\rangle$,
$|101111\rangle$,
$|111001\rangle$,
$|111100\rangle$,

$|111101\rangle$, and
$|111111\rangle$.
Therefore, there is an equal likelihood of measuring the first three qubits as $|000\rangle$, $|010\rangle$, $|101\rangle$, or $|111\rangle$. For these, the inner product with c is 0: $\langle 0, c\rangle = \langle 2, c\rangle = \langle 5, c\rangle = \langle 7, c\rangle = 0$. The fact that $\langle 2, c\rangle = 0$ indicates that the second bit of c is 0. The fact that $\langle 5, c\rangle = 0$ indicates that the exclusive-or of the first bit of c and the third bit of c equals 0; therefore, they are either both 0 or 1. Since $f(x)$ is not a one-to-one function, c cannot be 0. Therefore, c must be 5. The following code automatically generates the matrix U:

```
class simon {
    public static final int SIZE=64;

    public static int f(int x)
    {
        if (x == 0)
            return 4;
        else if (x == 1)
            return 1;
        else if (x == 2)
            return 5;
        else if (x == 3)
            return 7;
        else if (x == 4)
            return 1;
        else if (x == 5)
            return 4;
        else if (x == 6)
            return 7;
        else if (x == 7)
            return 5;
    }

    public static void main(String args[])
    {
```

```
        int[][] matrix = new int[SIZE][SIZE];
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                matrix[i][j] = 0;
            }
        }
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                int x = (j>>3)&7;
                int y = j&7;
                int xprime=x;
                int yprime=y^f(x);
                int row = (xprime<<3)|yprime;
                int col = j;
                matrix[row][col] = 1;
            }
        }
        System.out.println("Define U " + SIZE);
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                System.out.print("+matrix[i][j]);
                if (j != SIZE-1)
                    System.out.print(" ");
            }
            if (i != SIZE-1)
                System.out.println(" ");
        }
    }
}
```

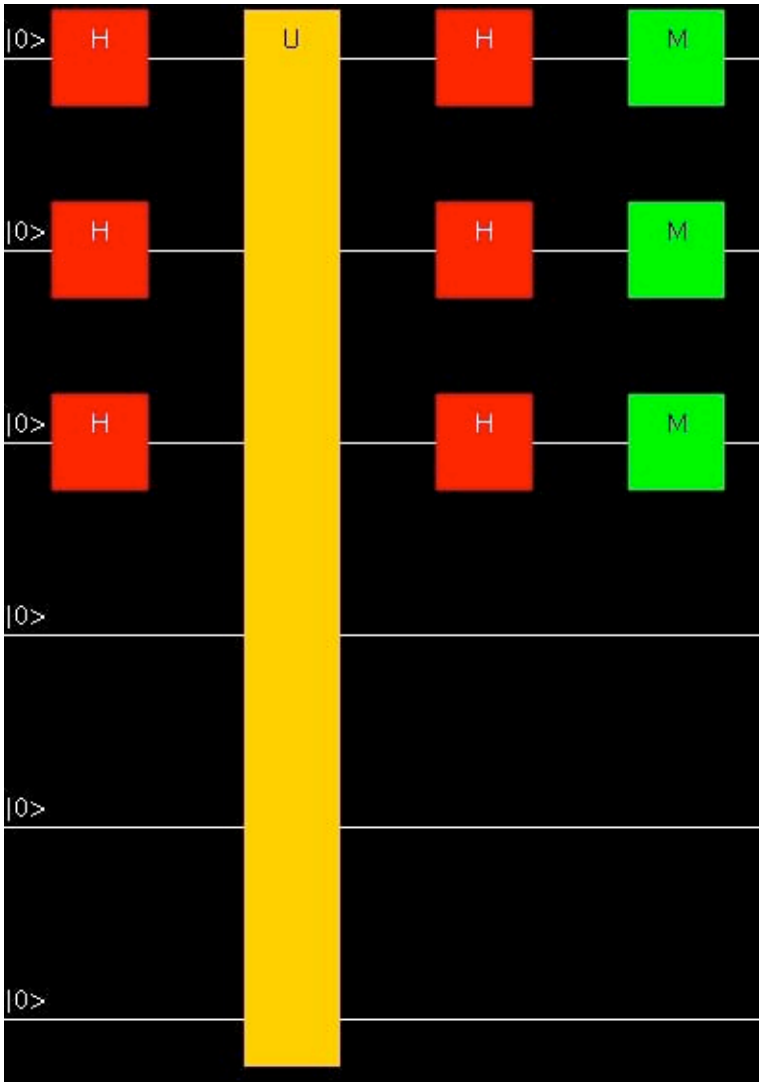


Figure 8.8: Circuit for Simon's Periodicity Algorithm

8.3.4 Grover's Algorithm

Given an unordered array of N elements, Grover's Algorithm finds a particular element in $N^{1/2}$ raised to the power $1/2$ steps. In other words, given a function whose domain is any natural number between 0 and $2^n - 1$, where n is the number of quantum bits, find x_0, where $f(x) = 1$ if $x = x_0$, and $f(x) = 0$ if $x \neq x_0$. In this example, $f(x)$ *picks out* $x_0 = 5$. Prior to measurement of the top three quantum bits, there is an equal superposition of all quantum states of this four-qubit system from $|0\rangle$ to $|15\rangle$, each having a probability of $(6.25\%)^2$, except for $|10\rangle$ and $|11\rangle$, which each have a probability of $(68.75\%)^2 = 47.3\%$. Clearly, the states we are most likely to

measure are $|1010\rangle$ or $|1011\rangle$. In both cases, the first three qubits are $101_2 = 5_{10}$. Five is the number we are looking for. The quantum circuit file is shown in Figure 7.1.

The following code automatically generates the two matrices U and V needed for Grover's Algorithm:

```
class grover {
    public static final int SIZE=16;

    public static int f(int x)
    {
        if (x == 0)
            return 0;
        else if (x == 1)
            return 0;
        else if (x == 2)
            return 0;
        else if (x == 3)
            return 0;
        else if (x == 4)
            return 0;
        else if (x == 5)
            return 1;
        else if (x == 6)
            return 0;
        else if (x == 7)
            return 0;
    }

    public static void main(String args[])
    {
        int[][] matrix = new int[SIZE][SIZE];
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                matrix[i][j] = 0;
```

```
    }
}
for (int i = 0; i < SIZE; i++) {
    for (int j = 0; j < SIZE; j++) {
        int x = (j>>1)&7;
        int y = j&1;
        int xprime=x;
        int yprime=y^f(x);
        int row = (xprime<<1)|yprime;
        int col = j;
        matrix[row][col] = 1;
    }
}
System.out.println("Define U " + SIZE);
for (int i = 0; i < SIZE; i++) {
    for (int j = 0; j < SIZE; j++) {
        System.out.print("+matrix[i][j]);
        if (j != SIZE-1)
            System.out.print(" ");
    }
    if (i != SIZE-1)
        System.out.println(" ");
}
System.out.println();
System.out.println("Define V "+SIZE/2);
double A = 1.0 / ((double)SIZE/2.0);
for (int i = 0; i < SIZE/2; i++) {
    for (int j = 0; j < SIZE/2; j++) {
        if (i == j)
            System.out.print("+(-1+2*A));
        else
            System.out.print("+2*A);
        if (j < SIZE/2-1)
            System.out.print(" ");
```

```
                }
                System.out.println();
            }
        }
}
```

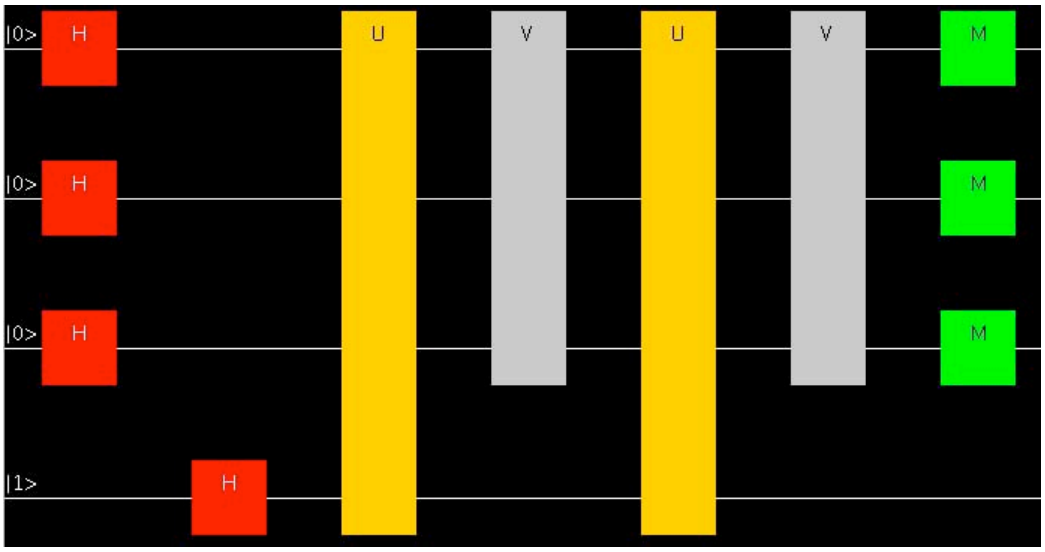


Figure 8.9: Circuit for Grover's Algorithm

8.3.5 Shor's Algorithm

Shor's Algorithm reduces the problem of factoring to the problem of finding the period of a function $f(x) = a^x \bmod N$, where N is the number to be factored, and a is a random integer that is less than N but does not have a nontrivial factor in common with N. The algorithm consists of a quantum part and a classical part. The quantum part of the algorithm determines the period r, and the classical part of the algorithm uses Euclid's algorithm to determine the factors from r. If r is negative, a different value of a must be chosen. In addition, $a^{r/2} \neq -1 \bmod N$. In this example, $a = 7$, and $N = 15$. At the end of the algorithm, prior to measurement of the upper three qubits, there is an equal superposition of

$|1\rangle = |0000001\rangle$,

$|33\rangle = |0100001\rangle$,

$|65\rangle = |1000001\rangle$, and

$|97\rangle = |1100001\rangle$,

which means that there is an equal probability of measuring the upper three qubits as 000, 010,

100, and 110. If the upper three qubits are measured as $110_2 = 6_{10}$, this indicates that 6 is a multiple of $2^3/r$. In other words, $6 = \lambda 2^3/r$. Therefore, $6/2^3 = \lambda/r$. We reduce $6/(2^3) = 6/8$ to an irreducible fraction 3/4 = λ/r, which indicates that $r = 4$. We then plug r into a formula to calculate the factors: $GCD(a^{r/2+1},N)$ and $GCD(a^{r/2-1}, N)$. Therefore, the factors are $GCD(7^2 + 1, 15) = GCD(50, 15) = 5$ and $GCD(7^2 - 1, 15) = GCD(45, 15) = 3$.

```
Define N 7
Define U 128
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Define Phi0
0 0 0 0 0 0 0
Define Transform1
H H H I I I I
Define Transform2
U
Define Transform3
I I I M M M M
Define Transform4
S1 I S2 I I I I
Define Transform5
I I H I I I I
Define Transform6
I UCSP1 UCSP2 I I I I
Define Transform7
I H I I I I I
Define Transform8
UCTP1 I UCTP2 I I I I
Define Transform9
UCSP1 UCSP2 I I I I I
Define Transform10
H I I I I I I
Define Transform11
M M M I I I I
```

The following code automatically generates the matrix U:

```
class shor {
    public static void main(String args[])
    {
        int a = 7;
        int N = 15;
        int n = 3;  // x-register (upper register)
        int m = 4;  // y-register (lower register)
        int SIZE=pow(2,n+m);
        int[][] matrix = new int[SIZE][SIZE];
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                matrix[i][j] = 0;
            }
        }
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                int x = (j>>m)&(pow(2,n)-1);
                int y = j&(pow(2,m)-1);
                int xprime=x;
                int yprime = y ^ (pow(a,x) % N);
                int row = (xprime<<m)|yprime;
                int col = j;
                matrix[row][col] = 1;
            }
        }
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                System.out.print("+matrix[i][j]);
                if (j != SIZE-1) {
                    System.out.print(" ");
                }
            }
```

```
            if (i != SIZE-1) {
                System.out.println();
            }
        }
    }
}
```

The following code automatically calculates the factors from the measurement of the upper three qubits:

```
void determine_factors(int a, int N, int x, int xbits)
{
    int numerator = x;
    int denominator = pow(2,xbits);
    while (true) {
        int divisor = gcd(numerator,denominator);
        if (divisor == 1)
            break;
        numerator /= divisor;
        denominator /= divisor;
    }
    int factor1 = gcd(pow(a,denominator/2)+1,N);
    int factor2 = gcd(pow(a,denominator/2)-1,N);
    if (factor1 * factor2 != N)
        System.out.println("Try again.");
}
```

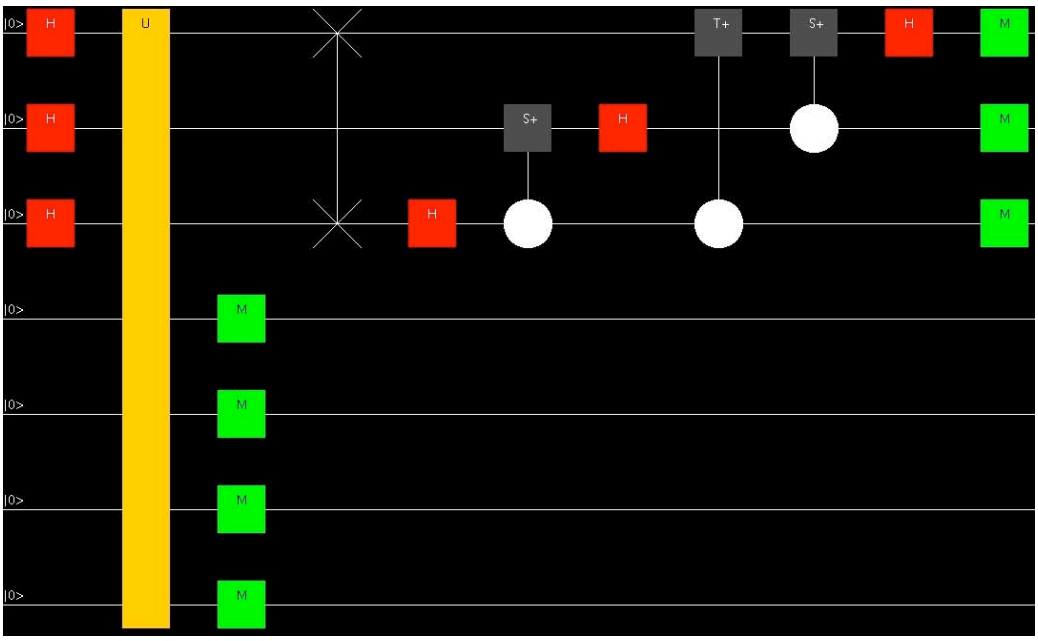


Figure 8.10: Circuit for Shor's Algorithm

8.3.6 Shor's Algorithm (Alternate Version)

This circuit is an alternate version of Shor's Algorithm.

```
Define N 7
Define U 128
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
...
Define Phi0
0 0 0 0 0 0 0
Define Transform1
H I I I I I I
Define Transform2
UCS1 UCS2 I I I I I
Define Transform3
UCT1 I UCT2 I I I I
Define Transform4
I H I I I I I
```

```
Define Transform5
I UCS1 UCS2 I I I I
Define Transform6
I I H I I I I
Define Transform7
S1 I S2 I I I I
Define Transform8
U
Define Transform9
S1 I S2 I I I I
Define Transform10
I I H I I I I
Define Transform11
I UCSP1 UCSP2 I I I I
Define Transform12
I H I I I I I
Define Transform13
UCTP1 I UCTP2 I I I I
Define Transform14
UCSP1 UCSP2 I I I I I
Define Transform15
H I I I I I I
Define Transform16
M M M I I I I
```

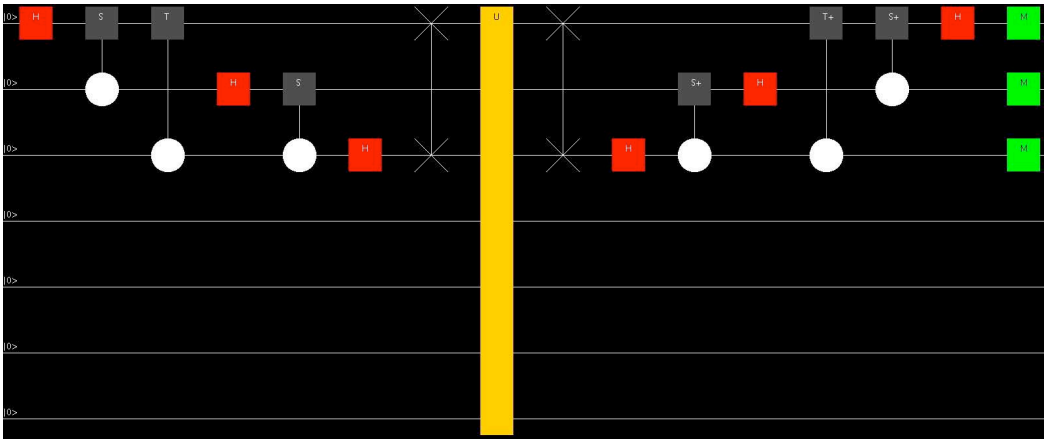


Figure 8.11: Alternate Circuit for Shor's Algorithm

8.4 Quantum Error Correction Schemes

8.4.1 Single Bit-Flip Error

This $(3, 1)$ repetition code circuit corrects a single bit-flip error. In this example, an X gate is applied to the upper qubit, resulting in a bit-flip error. This causes the syndrome qubits (the middle qubit and the lower qubit) to take on the value $|1\rangle$. This causes the Toffoli gate to flip the upper qubit, correcting the error.

```
Define N 3
Define Phi0
1 0 0
Define Transform1
Control Target I
Define Transform2
Control I Target
Define Transform3
X I I
Define Transform4
Control I Target
Define Transform5
Control Target I
Define Transform6
UT1 UT2 UT3
```

8.4.2 Single Phase-Flip Error

This $(3, 1)$ repetition code circuit corrects a single phase-flip error. The H gates change a phase-flip error to a bit-flip error. In this example, a Z gate is applied to the upper qubit. This causes the syndrome qubits (the middle qubit and the lower qubit) to take on the value $|1\rangle$. This causes the Toffoli gate to apply a bit-flip to the upper qubit, correcting the error.

```
Define N 3
Define Phi0
1 0 0
Define Transform1
Control Target I
Define Transform2
Control I Target
Define Transform3
H H H
Define Transform4
Z I I
Define Transform5
H H H
Define Transform6
Control I Target
Define Transform7
Control Target I
Define Transform8
UT1 UT2 UT3
```

8.4.3 Single Bit-Flip Error or Phase-Flip Error

This Shor $(9, 1, 3)$ error correcting code can correct a single bit-flip or phase-flip error. In this example, an X gate is applied to the upper qubit. This causes the second and third qubits to take on the value $|1\rangle$. Since these two qubits are the control bits of a Toffoli gate, the Toffoli gate flips the upper qubit, correcting the error.

```
Define N 9
Define Phi0
1 0 0 0 0 0 0 0 0
Define Transform1
Control I I Target I I I I I
Define Transform2
Control I I I I I Target I I
Define Transform3
H I I H I I H I I
Define Transform4
Control Target I Control Target I Control Target I
Define Transform5
Control I Target Control I Target Control I Target
Define Transform6
X I I I I I I I I
Define Transform7
Control I Target Control I Target Control I Target
Define Transform8
Control Target I Control Target I Control Target I
Define Transform9
UT1 UT2 UT3 UT1 UT2 UT3 UT1 UT2 UT3
Define Transform10
H I I H I I H I I
Define Transform11
Control I I I I I Target I I
Define Transform12
Control I I Target I I I I I
Define Transform13
UT1 I I UT2 I I UT3 I I
```

In the next example circuit, a Z gate is applied to the upper qubit. H gates convert the phase-flip error to a bit-flip error. The syndrome qubits (the fourth and seventh qubits) take on the value $|1\rangle$. Since the syndrome qubits are the control bits of a Toffoli gate, they cause the Toffoli gate to flip the upper qubit, correcting the error.

```
Define N 9
Define Phi0
1 0 0 0 0 0 0 0 0
Define Transform1
Control I I Target I I I I I
Define Transform2
Control I I I I I Target I I
Define Transform3
H I I H I I H I I
Define Transform4
Control Target I Control Target I Control Target I
Define Transform5
Control I Target Control I Target Control I Target
Define Transform6
Z I I I I I I I I
Define Transform7
Control I Target Control I Target Control I Target
Define Transform8
Control Target I Control Target I Control Target I
Define Transform9
UT1 UT2 UT3 UT1 UT2 UT3 UT1 UT2 UT3
Define Transform10
H I I H I I H I I
Define Transform11
Control I I I I I Target I I
Define Transform12
Control I I Target I I I I I
Define Transform13
UT1 I I UT2 I I UT3 I I
```

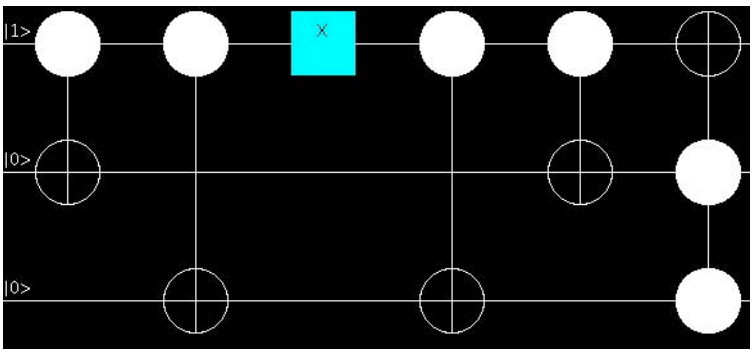


Figure 8.12: Circuit for Bit Flip

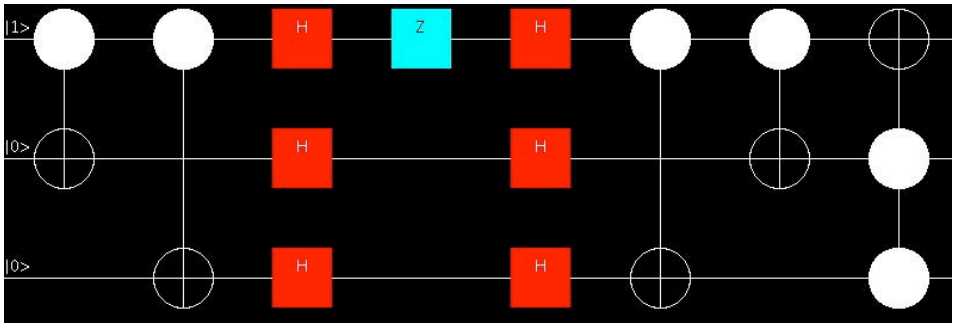


Figure 8.13: Circuit for Phase Flip

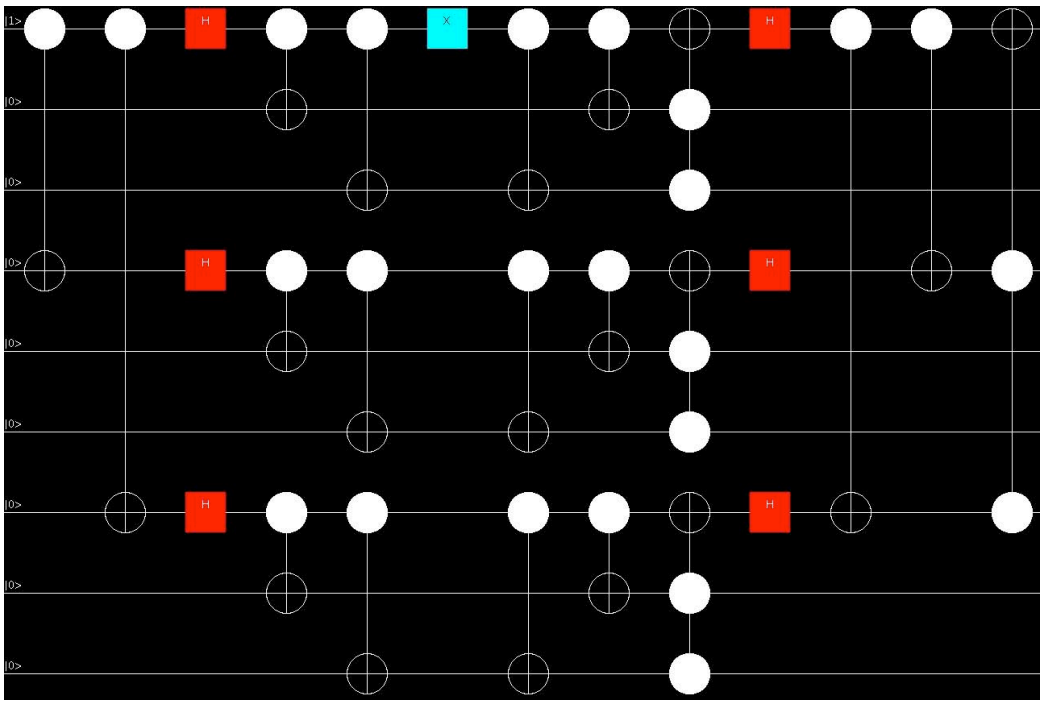


Figure 8.14: Circuit for Bit Flip or Phase Flip

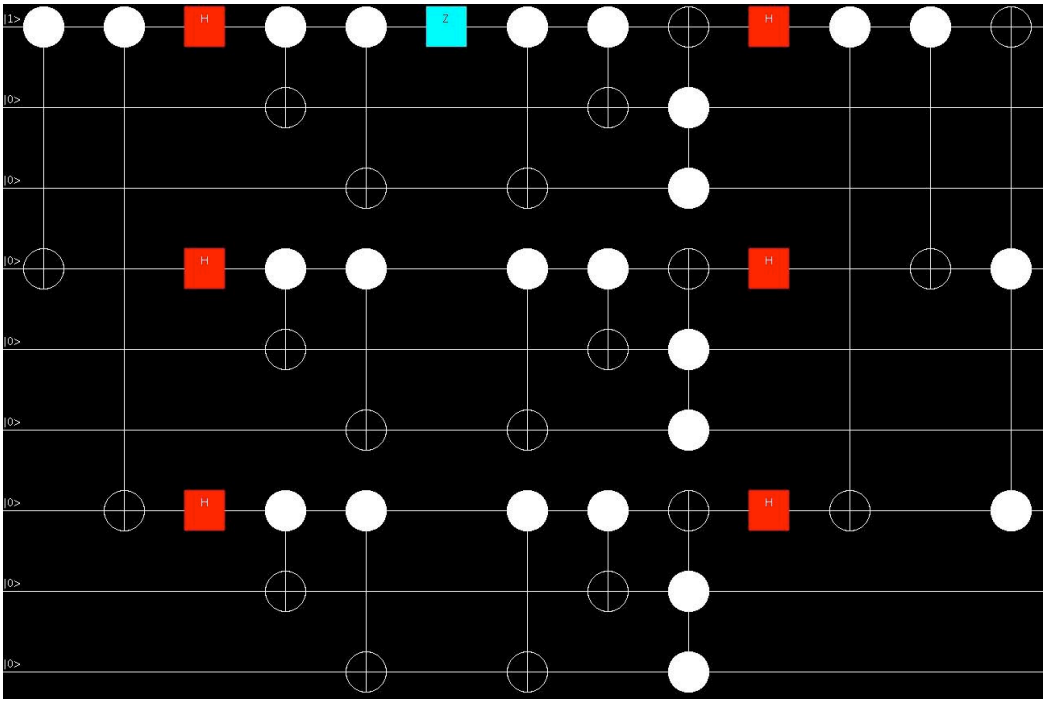


Figure 8.15: Circuit for Bit Flip or Phase Flip

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 9:
Analysis of Error Correction Simulations

The literature on quantum error-correction is vast and can be daunting at times. Luckily, Professor Marek Perkowski of Portland State University has posted a set of lecture slides titled *Shor's 9-Qubit Error Correction Code* to his course web site, which provide a very clear explanation of a few error correcting codes, along with mathematical analysis [28]. Based on these slides, a set of experiments were developed to validate the design flow and to more fully understand a set of error correction schemes. The experiments and findings are described below:

Experiment 1: (3,1) Repetition Code for correcting a single bit-flip error: bit-flip applied to uppermost qubit
The initial state of the circuit was $|4\rangle$: uppermost qubit $|1\rangle$, all others $|0\rangle$. The final state of the circuit was $|7\rangle$: all qubits $|1\rangle$. This circuit corrected a bit-flip error applied to the uppermost (data) qubit. The bit-flip caused the syndrome qubits to be flipped from $|0\rangle$ to $|1\rangle$, which caused the Toffoli gate to flip the uppermost (data) qubit from $|0\rangle$ back to its correct value of $|1\rangle$.

Experiment 2: (3,1) Repetition Code for correcting a single bit-flip error: bit-flip applied to middle qubit
The initial state of the circuit was $|4\rangle$: Upper qubit $|1\rangle$, all others $|0\rangle$. The final state of the circuit was $|6\rangle$: Upper and middle qubits $|1\rangle$, lower qubit $|0\rangle$. Since the bit-flip was applied to a syndrome qubit, that qubit was correctly flipped from $|0\rangle$ to $|1\rangle$. The bit-flip did not affect the data qubit. This circuit does not include recovery logic to flip the syndrome qubits, although it would be easy to add.

Experiment 3: (3,1) Repetition Code for correcting a single phase-flip error: phase-flip applied to uppermost qubit
The initial state of the circuit was $|4\rangle$: uppermost qubit $|1\rangle$, all others $|0\rangle$. The final state of the circuit was $|7\rangle$: All qubits $|1\rangle$. This circuit corrected a phase-flip error applied to the uppermost qubit. The phase-flip caused the syndrome qubits to be flipped from $|0\rangle$ to $|1\rangle$, which caused the Toffoli gate to flip the data (uppermost) qubit from $|0\rangle$ back to its correct value of $|1\rangle$.

Note that this circuit is very similar to the $(3, 1)$ Repetition Code circuit for correcting a single bit-flip error, except for the additional H gates. This circuit converts a phase-flip error to a bit-flip error, exploiting the fact that $H \cdot Z \cdot H = X$.

Experiment 4: (3,1) Repetition Code for correcting a single phase-flip error: phase-flip applied to uppermost qubit

The initial state of the circuit was $|0\rangle$: All qubits $|0\rangle$. The final state of the circuit was $|3\rangle$: Upper qubit $|0\rangle$, all others $|1\rangle$. The phase-flip on the data qubit caused the syndrome qubits to be flipped from $|0\rangle$ to $|1\rangle$, which caused the Toffoli gate to flip the data qubit from $|1\rangle$ back to its correct value of $|0\rangle$.

Experiment 5: (3,1) Repetition Code for correcting a single phase-flip error: phase-flip applied to middle qubit

The initial state of the circuit was $|4\rangle$: uppermost qubit $|1\rangle$, all others $|0\rangle$. The final state of the circuit was $|6\rangle$: upper and middle qubits $|1\rangle$, lower qubit $|0\rangle$. Since the phase-flip was applied to a syndrome (non-data) qubit, the syndrome qubit was correctly flipped from $|0\rangle$ to $|1\rangle$.

Experiment 6: (3,1) Repetition Code for correcting a single phase-flip error: phase-flip applied to lowermost qubit

The initial state of the circuit was $|4\rangle$: uppermost qubit $|1\rangle$, all others $|0\rangle$. The final state of the circuit was $|5\rangle$: upper and lower qubits $|1\rangle$, middle qubit $|0\rangle$. Since the phase-flip occurred on a syndrome qubit, that qubit was correctly flipped from $|0\rangle$ to $|1\rangle$. The error did not occur on the data qubit. This circuit does not include recovery circuitry to flip the syndrome qubits, although it would be easy to add.

Experiment 7: Shor's [9,1,3] Code: Bit-flip applied to uppermost qubit

Shor's $[9, 1, 3]$ Code can correct any arbitrary single-qubit error.

The initial state of the circuit was $|256\rangle$: uppermost qubit $|1\rangle$, all others $|0\rangle$. The final state of the circuit was $|448\rangle$: uppermost, 2^{nd}, and 3^{rd} qubits $|1\rangle$, all others $|0\rangle$. The bit-flip of the data qubit caused the syndrome qubits to be flipped from $|0\rangle$ to $|1\rangle$, which caused the Toffoli gate to flip the data qubit from $|0\rangle$ back to its correct value of $|1\rangle$.

Experiment 8: Shor's [9,1,3] Code: Phase-flip applied to uppermost qubit

The initial state of the circuit was $|256\rangle$: uppermost qubit $|1\rangle$, all others $|0\rangle$. The final state of the circuit was $|292\rangle$: uppermost, 4^{th}, and 7^{th} qubits $|1\rangle$, all others $|0\rangle$. The phase-flip applied to the uppermost (data) qubit caused the 4^{th} and 7^{th} (syndrome) qubits to be flipped from $|0\rangle$ to $|1\rangle$, which caused the Toffoli gate to flip the data qubit from $|0\rangle$ back to its correct value of $|1\rangle$.

Experiment 9: Shor's [9,1,3] Code: Bit-flip applied to uppermost qubit
The initial state of the circuit was $|0\rangle$: all qubits $|0\rangle$. The final state of the circuit was $|192\rangle$: 2^{nd} and 3^{rd} qubits $|1\rangle$, all others $|0\rangle$. The bit-flip of the data qubit caused the syndrome qubits to be flipped from $|0\rangle$ to $|1\rangle$, which caused the Toffoli gate to flip the data qubit from $|1\rangle$ back to its correct value of $|0\rangle$.

Experiment 10: Shor's [9,1,3] Code: Phase-flip applied to uppermost qubit
The initial state of the circuit was $|0\rangle$: uppermost qubit $|1\rangle$, all others $|0\rangle$. The final state of the circuit was $|36\rangle$: 4^{th} and 7^{th} qubits $|1\rangle$, all others $|0\rangle$. The phase-flip applied to the uppermost (data) qubit caused the 4^{th} and 7^{th} (syndrome) qubits to be flipped from $|0\rangle$ to $|1\rangle$, which caused the Toffoli gate to flip the data qubit from $|1\rangle$ back to its correct value of $|0\rangle$.

Experiment 11: Shor's [9,1,3] Code: Bit-flip applied to 4^{th} qubit.
The initial state of the circuit was $|256\rangle$: uppermost qubit $|1\rangle$, all others $|0\rangle$. The final state of the circuit was $|280\rangle$: uppermost, 5^{th}, and 6^{th} qubits $|1\rangle$, all others zero. This makes sense because the bit-flip was applied to the 4^{th} qubit. The 5^{th} and 6^{th} qubits were correctly changed from $|0\rangle$ to $|1\rangle$, causing the Toffoli gate to correct the error, restoring the 4^{th} qubit to its correct value of $|0\rangle$.

Experiment 12: Shor's [9,1,3] Code: Phase-flip applied to 4^{th} qubit.
The initial state of the circuit was $|256\rangle$: uppermost qubit $|1\rangle$, all others $|0\rangle$. The final state of the circuit was $|288\rangle$: uppermost and 4^{th} qubits $|1\rangle$, all others $|0\rangle$. This makes sense because the phase-flip was applied to the fourth qubit. In this case, the error occurred on a non-data (syndrome) qubit, which was (correctly) flipped. The circuit is designed to correct either a phase-flip or bit-flip on the data qubit. Although it does not have recovery circuitry to flip non-data (syndrome) qubits, this functionality could easily be implemented with a small number of additional quantum gates.

Experiment 13: Shor's [9,1,3] Code: No error applied
The initial state of the circuit was as follows: uppermost qubit $|1\rangle$, all others $|0\rangle$. The final state of the circuit was the same as the initial state, as expected, since no error was applied.

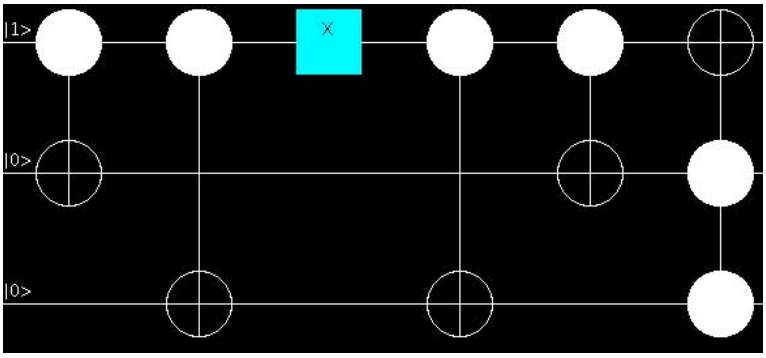


Figure 9.1: Experiment 1

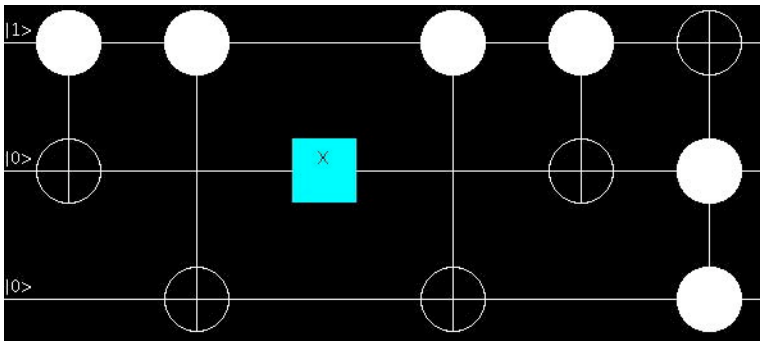


Figure 9.2: Experiment 2

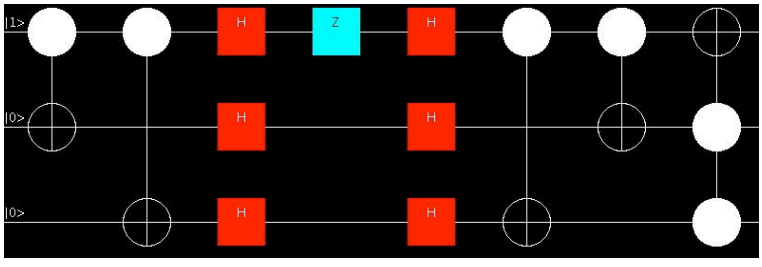


Figure 9.3: Experiment 3

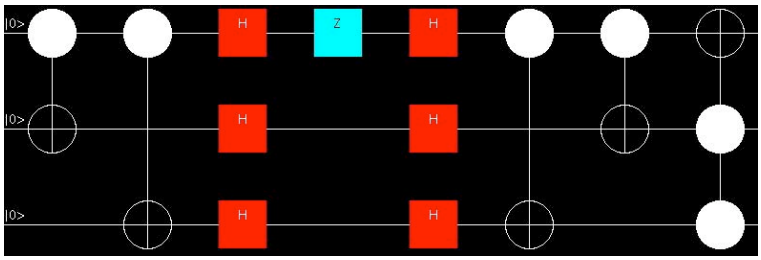


Figure 9.4: Experiment 4

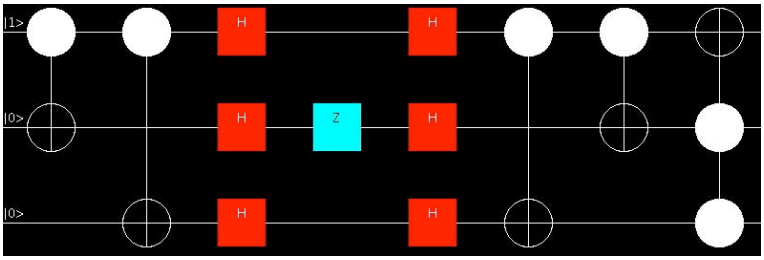


Figure 9.5: Experiment 5

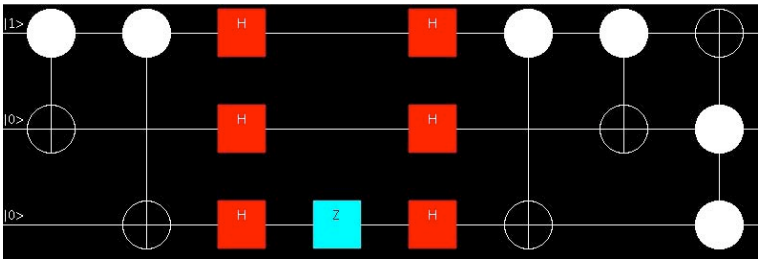


Figure 9.6: Experiment 6

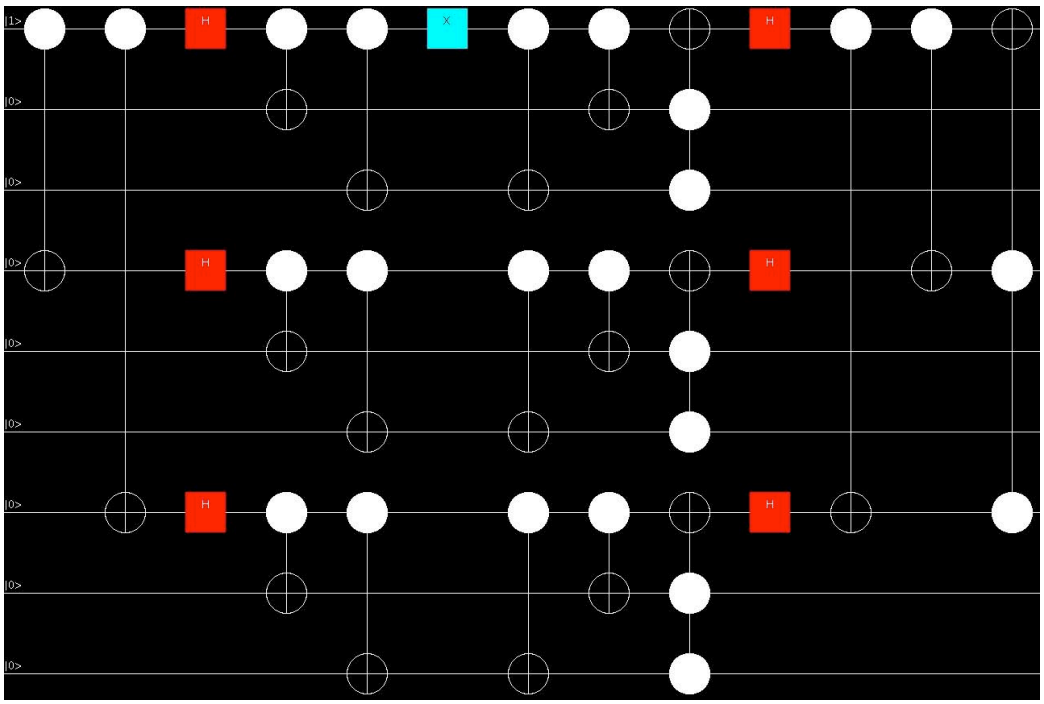


Figure 9.7: Experiment 7

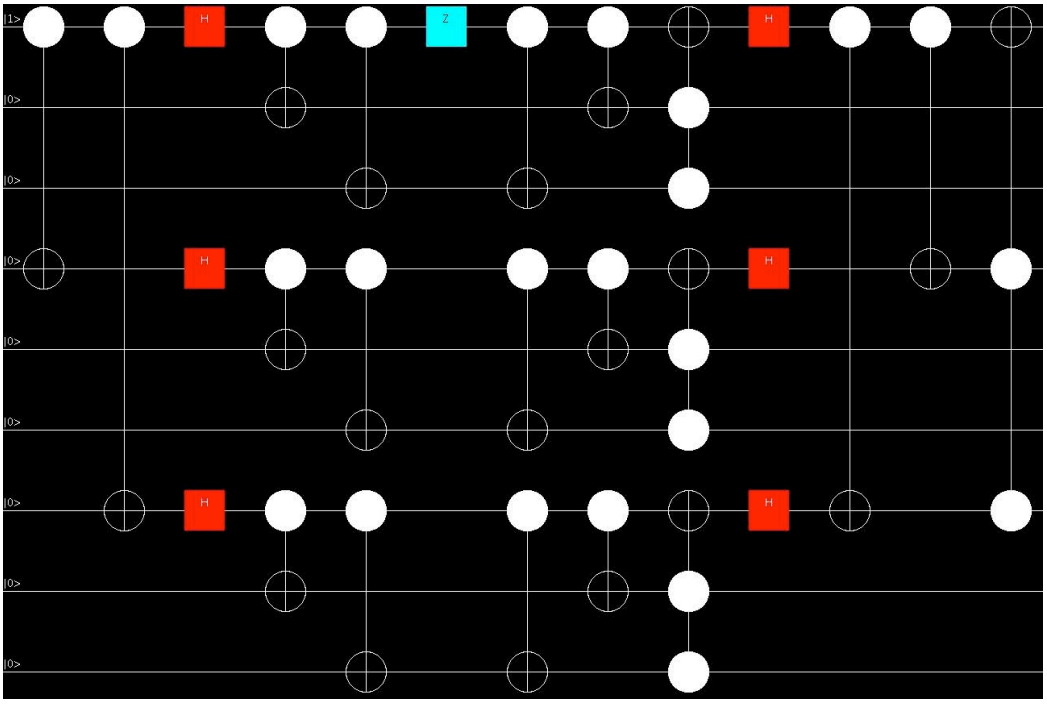


Figure 9.8: Experiment 8

CHAPTER 10:
Conclusion and Future Work

This design flow will be used in the next offering of the Quantum Computing course at NPS as a teaching tool for understanding quantum algorithms. We are considering releasing the source code to the public eventually, once the tools have been refined sufficiently. The design flow has a small code base and offers a useful set of features. It has several unique advantages over currently available quantum computer simulators, many of which are closed-source.

This work on designing quantum circuits, simulating quantum circuits, understanding quantum algorithms, and understanding quantum error correction schemes lays the groundwork necessary for designing large-scale, fault-tolerant quantum computer architectures. Quantum computers will require large numbers of quantum bits, which must be put together in a manner that results in a winning design in which the benefits of quantum parallelism offset the costs of quantum error correction. To achieve this goal it will be necessary to use ideas from classical computer architecture while simultaneously avoiding the pitfall of remaining stuck in the mindset of the classical paradigm. Computer architecture considers the components of a computer and how those components will interact with each other. What are the components of a large-scale quantum computer design, and how will these components interact with each other? Design tools will be needed to design them. What are the best design tools for quantum computer architectures, and can they be improved? Methods will be needed for evaluating and validating large-scale designs. This is challenging because it is not possible to simulate a large-scale design of a quantum computer on a classical computer due to the computing resource requirements, which grow exponentially in the size of the quantum circuit [14]. What are the metrics of success? What exactly does it mean to be a winning design?

We did not study every possible quantum error correction scheme ever devised. Future work will explore additional error correction schemes. Future work will also add features to the design flow to make it possible to automatically generate larger quantum circuits for Shor's Algorithm for factoring arbitrary numbers, using arbitrary parameters. The design tool and simulator should be integrated into a single application. The design flow should integrate the code for generating the U matrices into the quantum circuit designer program, rather than running separate programs for generating these matrices. The design flow should also decompose the U matrices into primitive quantum gates. The designer program should also incorporate

undo and zoom features. The design flow should also make it possible to evaluate how a given quantum algorithm will map to a particular physical implementation, taking into account the underlying properties of the physical technology used to build the quantum bits.

Last but not least, research should be undertaken to develop other quantum algorithms that would be useful to Computer Scientists. While predicting the future is difficult, there has been much recent progress in the development of quantum algorithms [45] and physical implementations [46]. However, no physical implementation technology currently meets the minimum threshold of reliability such that a large-scale, fault-tolerant quantum computer can be built, even with quantum error correction [47]. To see a detailed timeline of the evolution of quantum computing, see [14] and [48].

List of References

[1] D. Stick, W. K. Hensinger, S. Olmschenk, M. J. Madsen, K. Schwab, and C. Monroe, "Ion trap in a semiconductor chip," *Nature Phys.*, vol. 2, p. 36, 2006. [Online]. Available: http://www.citebase.org/abstract?id=oai:arXiv.org:quant-ph/0601052

[2] K. K. Berggren, "Quantum computing with superconductors," *Proc. IEEE*, vol. 92, no. 10, pp. 1630–1638, Oct 2004.

[3] R. Landauer, "Information is physical," *Phys. Today*, vol. 44, pp. 23–29, May 1991.

[4] C. E. Shannon and W. Weaver, *A Mathematical Theory of Communication*. Champaign, IL, USA: University of Illinois Press, 1963.

[5] C. H. Bennett, "Logical reversibility of computation," *IBM J. Res. Develop.*, Nov 1973.

[6] C. H. Bennett, "The thermodynamics of computation–a review," *Int. J. Theoretical Phys.*, vol. 21, no. 12, pp. 905–940, Dec 1982. [Online]. Available: http://dx.doi.org/10.1007/BF02084158

[7] E. Fredkin and T. Toffoli, "Conservative logic," pp. 47–81, 2002.

[8] C. H. Bennett and P. W. Shor, "Quantum information theory," *IEEE Tran. Inform. Theory*, vol. 44, no. 6, pp. 2724–2742, Oct 1998. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=366851&isnumber=8405

[9] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Rev.*, vol. 41, no. 2, pp. 303–332, 1999. [Online]. Available: http://portal.acm.org/ft_gateway.cfm?id=325514&type=external&coll=Portal&dl=GUIDE&CFID=75857286&CFTOKEN=75571989

[10] L. M. Vandersypen *et al.*, "Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance," *Nature*, vol. 414, no. 20/27, pp. 883–887, Dec 2001. [Online]. Available: http://arxiv.org/abs/quant-ph/0112176

[11] B. P. Lanyon *et al.*, "Experimental demonstration of Shor's algorithm with quantum entanglement," *Phys. Rev. Lett.*, vol. 99, p. 4, Dec 2007. [Online]. Available: http://arxiv.org/abs/0705.1398

[12] A. Politi, J. C. F. Matthews, and J. L. O'Brien, "Shor's quantum factoring algorithm on a photonic chip," *Science*, vol. 325, p. 1221, 2009. [Online]. Available: doi:10.1126/science.1173731

[13] L. K. Grover, "A fast quantum mechanical algorithm for database search," 1996. [Online]. Available: http://www.citebase.org/abstract?id=oai:arXiv.org:quant-ph/9605043

[14] T. S. Metodi and F. T. Chong, *Quantum Computing for Computer Architects (Synthesis Lectures on Computer Architecture)*. Morgan and Claypool Publishers, 2006.

[15] C. H. Bennett and G. Brassard, "Quantum cryptography: public key distribution and coin tossing," 1984.

[16] A. K. Ekert, "Quantum cryptography based on Bell's theorem," *Phys. Rev. Lett.*, vol. 67, no. 6, pp. 661–663, Aug 1991.

[17] C. H. Bennett, "Quantum cryptography using any two nonorthogonal states," *Phys. Rev. Lett.*, vol. 68, no. 21, pp. 3121–3124, May 1992.

[18] C. H. Bennett, G. Brassard, and N. D. Mermin, "Quantum cryptography without Bell's theorem," *Phys. Rev. Lett.*, vol. 68, no. 5, pp. 557–559, Feb 1992.

[19] Quantiki, "Quantiki," January 2010. [Online]. Available: http://www.quantiki.org/

[20] M. Peev *et al.*, "The SECOQC quantum key distribution network in Vienna," *New J. Phys.*, vol. 11, no. 7, p. 075001, Jul. 2009.

[21] R. Ursin *et al.*, "Space-quest: Experiments with quantum entanglement in space," 2008. [Online]. Available: http://www.citebase.org/abstract?id=oai:arXiv.org:0806.0945

[22] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[23] R. T. Perry, *The Temple of Quantum Computing*, www.toqc.com, Ed. online e-book, Apr 2006, vol. 1.1.

[24] A. Einstein, B. Podolsky, and N. Rosen, "Can quantum-mechanical description of physical reality be considered complete?" *Phys. Rev.*, vol. 47, no. 10, pp. 777–780, May 1935.

[25] R. P. Feynman, *Feynman Lectures on Computation*, A. J. Hey and R. W. Allen, Eds. Cambridge, MA, USA: Perseus Books, 2000.

[26] D. Deutsch, "Quantum theory, the Church-Turing principle and the universal quantum computer," *Proc. Roy. Soc. Lond.: Mathematical and Physical Sciences*, vol. 400, no. 1818, pp. 97–117, 1985.

[27] P. W. Shor, "Scheme for reducing decoherence in quantum computer memory," *Phys. Rev. Lett. A*, vol. 52, no. 4, pp. 2493–2496, Oct 1995.

[28] M. Perkowski, "Quantum computing," Apr 2007. [Online]. Available: http://www.ee.pdx.edu/~mperkows/CLASS_FUTURE/2005-quantum/2005-q-0018.error-models-9-bit-Shor.ppt

[29] D. P. DiVincenzo, "The physical implementation of quantum computation," pp. 1–9, Apr 2000. [Online]. Available: http://arxiv.org/abs/quant-ph/0002077

[30] L. M. Vandersypen *et al.*, "Experimental realization of an order-finding algorithm with an NMR quantum computer," *Phys. Rev. Lett.*, vol. 85, no. 25, pp. 5452–5455, Dec 2000. [Online]. Available: http://arxiv.org/abs/quant-ph/0007017

[31] W. S. Warren, N. Gershenfeld, and I. L. Chuang, "The usefulness of NMR quantum computing," *Science*, vol. 207, no. 5332, pp. 1688–1690, Sept 1997.

[32] B. D. Josephson, "The discovery of tunnelling supercurrents," *Mod. Phys. Rev.*, vol. 46, no. 2, pp. 251–254, Apr 1974.

[33] Y. Makhlin, G. Schon, and A. Shnirman, "Josephson-junction qubits," *Fortschritte Phys*, vol. 48, pp. 1043–1054, 2000.

[34] Y. Makhlin, G. Schon, and A. Shnirman, "Quantum state engineering with Josephson-junction devices," *Mod. Phys. Rev.*, vol. 73, p. 357, 2001. [Online]. Available: doi:10.1103/RevModPhys.73.357

[35] A. Imamoglu *et al.*, "Quantum information processing using quantum dot spins and cavity QED," *Phys. Rev. Lett.*, vol. 83, no. 20, pp. 4204–4207, Nov 1999.

[36] D. Loss and D. P. DiVincenzo, "Quantum computation with quantum dots," *Phys. Rev. Lett. A*, vol. 57, no. 1, pp. 120–126, Jan 1998.

[37] G. Burkard, H.-A. Engel, and D. Loss, "Spintronics and quantum dots for quantum computing and quantum communication," 2000. [Online]. Available: http://www.citebase.org/abstract?id=oai:arXiv.org:cond-mat/0004182

[38] T. B. Pittman, B. C. Jacobs, and J. D. Franson, "Quantum computing using linear optics," *John Hopkins APL Tech. Dig.*, vol. 25, p. 84, 2004. [Online]. Available: http://www.citebase.org/abstract?id=oai:arXiv.org:quant-ph/0406192

[39] "Linear optics quantum computing," Mar 2010. [Online]. Available: http://www.quantiki.org/wiki/index.php/Linear_optics_quantum_computation

[40] W. Paul, "Electromagnetic traps for charged and neutral particles," *Mod. Phys. Rev.*, vol. 62, no. 3, pp. 531–540, Jul 1990.

[41] J. I. Cirac and P. Zoller, "Quantum computations with cold trapped ions," *Phys. Rev. Lett.*, vol. 74, no. 20, pp. 4091–4094, May 1995.

[42] C. Monroe, D. M. Meekhof, B. E. King, W. M. Itano, and D. J. Wineland, "Demonstration of a fundamental quantum logic gate," *Phys. Rev. Lett.*, vol. 75, no. 25, pp. 4714–4717, Dec 1995.

[43] J. Cowie *et al.*, "A world wide number field sieve factoring record: On to 512 bits," in *ASIACRYPT '96: Proc. Int. Conf. Theory and Applicat. Cryptology Inf. Security*. London, UK: Springer-Verlag, 1996, pp. 382–394.

[44] D. Beckman, A. N. Chari, S. Devabhaktuni, and J. Preskill, "Efficient networks for quantum factoring," *Phys. Rev. A*, vol. 54, no. 2, pp. 1034–1063, Aug 1996.

[45] D. Bacon and W. van Dam, "Recent progress in quantum algorithms," vol. 53, no. 2, Feb 2010.

[46] M. Ross and M. Oskin, "Quantum computing," vol. 51, no. 7, July 2008.

[47] W. van Dam, "Quantum computing: In the 'Death Zone'?" vol. 3, no. 4, Apr 2007.

[48] (2010, Jan) Timeline of quantum computing. [Online]. Available: http://en.wikipedia.org/wiki/Timeline_of_quantum_computing

Referenced Authors

Initial Distribution List

1. Dudley Knox Library
   Naval Postgraduate School
   Monterey, CA

2. Dr. Peter Denning
   Naval Postgraduate School
   Monterey, CA

3. Dr. Ted Huffmire
   Naval Postgraduate School
   Monterey, CA

4. Dr. James Luscombe
   Naval Postgraduate School
   Monterey, CA

5. Dr. Karl van Bibber
   Naval Postgraduate School
   Monterey, CA